Azul Payara

Release 7.2.0

Table of Contents

General Info **1**

Documentation Overview 1

Release Strategy 1

Versioning Scheme 1

Examples 2

Supported Platforms 3

Supported Java Virtual Machines 3

Supported Operating Systems 3

Third Party Collaborations 4

Azul 4

Technical Documentation **6**

Payara Server Documentation 6

Payara Server Documentation 6

Documentation Conventions 7

Docker Images 9

General Administration 9

Overview of Payara Server Administration 9

General Runtime Administration 28

Using REST Interfaces to Administer Payara Server 56

Administering Domains 96

Administering the Virtual Machine for the Java Platform 123

Administration Console Features 129

Administering Thread Pools 135

Administering the Logging Service 141

Administering the Monitoring Service 189

Administering the Healthcheck Service 242

Administering the Request Tracing Service 250

Administering the Notification Service 256
Extended Notification Service Details 268
Administering Batch Jobs 291
Administering Database Connectivity 300
Administering EIS Connectivity 346
Administering HTTP Connectivity 363
Administering Concurrent Resources 409
Administering the Object Request Broker (ORB) 426
Administering the Jakarta Mail Service 430
Administering the Java Message Service (JMS) 435
Administering the Java Naming and Directory Interface (JNDI) Service 459
Administering Transactions 468
Administering Web Applications 483
Configuration Variables Reference 496
Mbeans Inventory 515
Deployment Planning 533
Overview of Payara Server Deployment Planning 533
Product Concepts 533
Planning Your Deployment 544
Deployment Checklist 562
Application Deployment 565
Overview of Payara Server Application Deployment 565
Deploying Applications 582
The `asadmin` Deployment Subcommands 622
Azul Payara Deployment Descriptor Files 626
Elements of the Azul Payara Deployment Descriptors 642
High Availability 805
High Availability in Payara Server 805

Enabling Centralized Administration of Payara Server Instances 813
Administering Payara Server Nodes 844
Administering Payara Server Clusters 862
Administering Deployment Groups 879
Administering the Domain Data Grid 887
Administering Payara Server Instances. 910
Administering Named Configurations 961
Configuring HTTP Load Balancing. 973
Configuring High Availability Session Persistence and Failover. 978
Configuring Java Message Service High Availability 991
RMI-IIOP Load Balancing and Failover 1004
Security Guide 1013
Overview 1013
Administering System Security 1013
Administering User Security 1055
Administering Message Security 1065
Administering Security in a High-Availability Environment 1080
Managing Administrative Security 1083
Running in a Secure Environment 1102
SSL Certificate Management 1114
Printing Certificate Data 1123
Command Reference 1126
Overview 1126
Instance 1128
Configuration 1128
Dotted Names 1129
Deployment Group 1134
Applications 1135

Auto-Naming. 1135
Logging 1137
Security 1138
Add-Instance-To-Deployment-Group 1139
Add-Library 1140
Add-Resources 1144
Add-To-Keystore. 1147
Add-To-Truststore 1148
Appclient 1150
Asadmin-Recorder-Enabled 1154
Asadmin 1155
Attach 1172
Backup-Domain 1173
Capture-Schema. 1175
Change-Admin-Password 1176
Change-Master-Broker 1178
Change-Master-Password. 1180
Clean-Jbatch-Repository 1181
Clear-Cache 1183
Collect-Diagnostics 1184
Collect-Log-Files 1187
Configure-Jms-Cluster 1190
Configure-Ldap-For-Admin 1192
Configure-Managed-Jobs 1193
Copy-Config 1194
Create-Admin-Object 1197
Create-Application-Ref 1199
Create-Auth-Realm. 1201

Create-Cluster. 1206
Create-Connector-Connection-Pool 1216
Create-Connector-Resource 1220
Create-Connector-Security-Map 1222
Create-Connector-Work-Security-Map 1224
Create-Context-Service 1226
Create-Custom-Resource 1228
Create-Deployment-Group. 1230
Create-Domain 1232
Create-File-User 1243
Create-Http-Listener 1244
Create-Http-Redirect 1247
Create-Http 1248
Create-Iiop-Listener 1250
Create-Instance 1252
Create-Jacc-Provider 1260
Create-Javamail-Resource 1262
Create-Jdbc-Connection-Pool. 1264
Create-Jdbc-Resource 1273
Create-Jms-Host 1275
Create-Jms-Resource 1276
Create-Jmsdest 1281
Create-Jndi-Resource 1285
Create-Jvm-Options 1287
Create-Jvm-Options 1291
Create-Local-Instance 1295
Create-Managed-Executor-Service. 1303
Create-Managed-Scheduled-Executor-Service 1307

Create-Managed-Thread-Factory 1309
Create-Message-Security-Provider 1311
Create-Module-Config 1314
Create-Network-Listener 1316
Create-Node-Config 1318
Create-Node-Docker 1321
Create-Node-Ssh 1323
Create-Password-Alias 1327
Create-Protocol-Filter 1329
Create-Protocol-Finder 1331
Create-Protocol 1332
Create-Resource-Adapter-Config 1333
Create-Resource-Ref 1334
Create-Service 1336
Create-Ssl 1344
Create-System-Properties 1347
Create-System-Properties 1348
Create-Threadpool 1350
Create-Transport 1352
Create-Virtual-Server 1354
Debug-Asadmin 1362
Delete-Admin-Object 1362
Delete-Application-Ref 1364
Delete-Auth-Realm 1366
Delete-Cluster 1367
Delete-Config-Property 1369
Delete-Config 1370
Delete-Connector-Connection-Pool 1372

Delete-Connector-Resource 1373
Delete-Connector-Security-Map 1375
Delete-Connector-Work-Security-Map 1376
Delete-Context-Service 1377
Delete-Custom-Resource 1379
Delete-Deployment-Group 1380
Delete-Domain 1382
Delete-File-User 1383
Delete-Http-Listener 1385
Delete-Http-Redirect 1386
Delete-Http 1387
Delete-Iiop-Listener 1388
Delete-Jacc-Provider 1392
Delete-Javamail-Resource 1393
Delete-Jdbc-Connection-Pool 1395
Delete-Jdbc-Resource 1396
Delete-Jms-Host 1397
Delete-Jms-Resource 1398
Delete-Jmsdest 1400
Delete-Jndi-Resource 1402
Delete-Jvm-Options 1403
Delete-Local-Instance 1406
Delete-Managed-Executor-Service 1408
Delete-Managed-Scheduled-Executor-Service 1410
Delete-Managed-Thread-Factory 1411
Delete-Message-Security-Provider 1412
Delete-Module-Config 1414
Delete-Network-Listener 1415

Delete-Node-Config 1417
Delete-Node-Docker 1419
Delete-Node-Ssh 1420
Delete-Password-Alias 1422
Delete-Protocol-Filter 1423
Delete-Protocol-Finder 1425
Delete-Protocol 1426
Delete-Resource-Adapter-Config 1427
Delete-Resource-Ref 1428
Delete-Service 1430
Delete-Ssl 1432
Delete-System-Property 1433
Delete-Threadpool 1434
Delete-Transport 1436
Delete-Virtual-Server 1437
Deploy-Remote-Archive 1438
Deploy 1448
Disable-Asadmin-Recorder 1458
Disable-Monitoring 1459
Disable-Phone-Home 1461
Disable-Secure-Admin-Internal-User 1462
Disable-Secure-Admin-Principal 1463
Disable-Secure-Admin 1465
Disable 1466
Enable-Asadmin-Recorder 1467
Enable-Monitoring 1468
Enable-Phone-Home 1471
Enable-Secure-Admin-Internal-User 1472

Enable-Secure-Admin-Principal 1474
Enable-Secure-Admin 1476
Enable 1478
Export-Sync-Bundle 1479
Export 1482
Flush-Connection-Pool 1484
Flush-Jmsdest 1485
Freeze-Transaction-Service 1487
Generate-Bash-Autocomplete 1488
Generate-Csr 1489
Generate-Encryption-Key 1491
Generate-Jvm-Report 1492
Generate-Self-Signed-Certificate 1495
Get-Active-Module-Config 1497
Get-Admin-Audit-Configuration 1499
Get-Asadmin-Recorder-Configuration 1500
Get-Aws-Config-Source-Configuration 1501
Get-Azure-Config-Source-Configuration 1502
Get-Cdieventbus-Notifier-Configuration 1503
Get-Config-Dir 1506
Get-Config-Ordinal 1507
Get-Config-Property 1508
Get-Datadog-Notifier-Configuration 1510
Get-Discord-Notifier-Configuration 1511
Get-Dynamodb-Config-Source-Configuration 1512
Get-Ejb-Invoker-Configuration 1513
Get-Email-Notifier-Configuration 1514
Get-Environment-Warning-Configuration 1515

Get-Eventbus-Notifier-Configuration 1516
Get-Fault-Tolerance-Configuration 1517
Get-Gcp-Config-Source-Configuration 1518
Get-Hashicorp-Config-Source-Configuration 1519
Get-Hazelcast-Configuration 1520
Get-Health 1522
Get-Healthcheck-Configuration 1523
Get-Jdbc-Config-Source-Configuration 1525
Get-Jms-Notifier-Configuration 1526
Get-Jmx-Monitoring-Configuration 1527
Get-Ldap-Config-Source-Configuration 1528
Get-Log-Notifier-Configuration 1529
Get-Metrics-Configuration. 1530
Get-Microprofile-Healthcheck-Configuration 1531
Get-Monitoring-Level 1532
Get-Monitoring-Service-Configuration 1534
Get-Newrelic-Notifier-Configuration 1535
Get-Notification-Configuration 1536
Get-Openapi-Configuration 1538
Get-Requesttracing-Configuration 1538
Get-Rest-Monitoring-Configuration 1539
Get-Slack-Notifier-Configuration 1541
Get-Snmp-Notifier-Configuration 1542
Get-Teams-Notifier-Configuration 1543
Get-Toml-Config-Source-Configuration 1544
Get 1545
Import-Sync-Bundle 1548
Install-Node-Ssh 1550

Install-Node 1554
Jms-Ping 1558
List-Admin-Objects 1559
List-Application-Refs 1560
List-Applications 1562
List-Auth-Realms 1564
List-Backups 1565
List-Batch-Job-Executions 1566
List-Batch-Job-Steps 1568
List-Batch-Jobs 1570
List-Batch-Runtime-Configuration 1572
List-Cache-Keys 1574
List-Caches 1576
List-Clusters 1577
List-Commands 1579
List-Connector-Connection-Pools 1584
List-Connector-Resources. 1585
List-Connector-Security-Maps 1586
List-Connector-Work-Security-Maps 1587
List-Containers 1589
List-Context-Services. 1590
List-Custom-Resources 1591
List-Deployment-Groups 1593
List-Domains. 1594
List-File-Groups 1595
List-File-Users 1597
List-Hazelcast-Cluster-Members 1598
List-Hazelcast-Members 1599

List-Healthcheck-Services 1600
List-Http-Listeners 1602
List-Iiop-Listeners 1603
List-Instances 1604
List-Jacc-Providers 1608
List-Javamail-Resources 1609
List-Jdbc-Connection-Pools 1610
List-Jdbc-Resources 1611
List-Jms-Hosts 1612
List-Jms-Resources 1614
List-Jmsdest 1615
List-Jndi-Entries 1617
List-Jndi-Resources 1618
List-Jobs 1619
List-Jvm-Options 1621
List-Keystore-Entries 1622
List-Libraries 1624
List-Log-Attributes 1625
List-Log-Levels 1627
List-Loggers 1629
List-Managed-Executor-Services 1630
List-Managed-Scheduled-Executor-Services 1632
List-Managed-Thread-Factories 1633
List-Message-Security-Providers 1634
List-Modules 1636
List-Network-Listeners 1648
List-Nodes-Config 1649
List-Nodes-Ssh 1652

List-Nodes 1654
List-Notifiers 1657
List-Password-Aliases 1658
List-Persistence-Types 1658
List-Phone-Home 1660
List-Protocol-Filters 1661
List-Protocol-Finders 1662
List-Protocols 1663
List-Requesttraces 1664
List-Resource-Adapter-Configs 1668
List-Resource-Refs 1669
List-Rest-Endpoints 1670
List-Secure-Admin-Internal-Users 1671
List-Secure-Admin-Principals 1673
List-Sub-Components 1674
List-Supported-Cipher-Suites 1676
List-System-Properties 1677
List-Threadpools 1679
List-Timers 1680
List-Transports 1681
List-Truststore-Entries 1682
List-Virtual-Servers 1684
List-Web-Context-Param 1685
List-Web-Env-Entry 1688
List 1690
Login 1694
Migrate-Timers 1696
Monitor 1697

Monitoring. 1701
Multimode. 1703
Notification-Configure 1705
Osgi-Shell 1706
Osgi 1708
Package-Appclient 1711
Ping-Connection-Pool 1713
Ping-Node-Ssh 1715
Print-Certificate 1717
Purge-Jbatch-Repository. 1719
Recover-Transactions 1721
Redeploy 1724
Remove-Expired-Certificates 1732
Remove-From-Keystore 1734
Remove-From-Truststore 1735
Remove-Instance-From-Deployment-Group 1737
Remove-Library 1738
Renew-Self-Signed-Certificates 1740
Resource 1742
Restart-Cluster 1742
Restart-Deployment-Group 1744
Restart-Domain 1747
Restart-Domains 1749
Restart-Hazelcast 1751
Restart-Http-Listeners 1752
Restart-Instance. 1754
Restart-Local-Instance 1756
Restart-Monitoring. 1760

Restore-Domain 1761
Rollback-Transaction 1763
Rotate-Log 1765
Set-Admin-Audit-Configuration 1766
Set-Amx-Enabled 1767
Set-Asadmin-Recorder-Configuration 1769
Set-Aws-Config-Source-Configuration 1770
Set-Azure-Config-Source-Configuration 1771
Set-Batch-Runtime-Configuration 1773
Set-Cdieventbus-Notifier-Configuration 1775
Set-Config-Cache 1776
Set-Config-Dir 1778
Set-Config-Ordinal 1779
Set-Config-Property 1780
Set-Datadog-Notifier-Configuration 1782
Set-Discord-Notifier-Configuration 1783
Set-Dynamodb-Config-Source-Configuration 1785
Set-Ejb-Invoker-Configuration 1786
Set-Email-Notifier-Configuration 1789
Set-Environment-Warning-Configuration 1790
Set-Eventbus-Notifier-Configuration 1791
Set-Fault-Tolerance-Configuration 1792
Set-Gcp-Config-Source-Configuration 1794
Set-Hashicorp-Config-Source-Configuration 1795
Set-Hazelcast-Configuration 1797
Set-Healthcheck-Configuration 1802
Set-Healthcheck-Service-Configuration 1804
Set-Jdbc-Config-Source-Configuration 1809

Set-Jms-Notifier-Configuration 1811
Set-Jmx-Monitoring-Configuration. 1813
Set-Ldap-Config-Source-Configuration 1815
Set-Log-Attributes 1817
Set-Log-File-Format 1820
Set-Log-Levels 1822
Set-Log-Notifier-Configuration 1823
Set-Metrics-Configuration. 1825
Set-Microprofile-Healthcheck-Configuration 1826
Set-Monitoring-Level 1828
Set-Monitoring-Service-Configuration 1830
Set-Network-Listener-Configuration 1831
Set-Newrelic-Notifier-Configuration. 1833
Set-Notification-Configuration 1834
Set-Openapi-Configuration 1836
Set-Requesttracing-Configuration 1838
Set-Rest-Monitoring-Configuration 1841
Set-Slack-Notifier-Configuration. 1842
Set-Snmp-Notifier-Configuration 1843
Set-Teams-Notifier-Configuration 1845
Set-Toml-Config-Source-Configuration 1846
Set-Web-Context-Param 1847
Set-Web-Env-Entry 1850
Set 1854
Setup-Ssh 1856
Show-Component-Status 1861
Start-Cluster 1862
Start-Database 1864

Start-Deployment-Group 1867
Start-Domain 1869
Start-Domains 1873
Start-Instance 1875
Start-Local-Instance 1879
Stop-All-Domains 1884
Stop-Cluster 1886
Stop-Database 1887
Stop-Deployment-Group 1889
Stop-Domain 1891
Stop-Domains 1893
Stop-Instance 1895
Stop-Local-Instance 1896
Undeploy 1899
Unfreeze-Transaction-Service 1900
Uninstall-Node-Ssh 1902
Uninstall-Node 1905
Unset-Web-Context-Param 1909
Unset-Web-Env-Entry 1911
Unset 1913
Update-Connector-Security-Map 1915
Update-Connector-Work-Security-Map 1916
Update-File-User 1918
Update-Node-Config 1919
Update-Node-Docker 1921
Update-Node-Ssh 1924
Update-Password-Alias 1927
Uptime 1928

Validate-Multicast 1929
Verify-Domain-Xml 1932
Version 1933
Extensions 1935
Server Extensions 1935
gRPC Support 1935
Payara Server Docker Image Overview 1946
Quick Start 1946
Other Distributions 1947
Exposed Ports 1947
Administration Details 1947
Installation Details 1948
Application Deployment 1948
Configuration 1949
Payara Insight 1954
Enabling the GUI 1954
Enabling Data Collection 1955
Enabling Data Aggregation (History) 1955
Graphical User Interface 1956
Diagnostics and Troubleshooting 2016
Diagnostics Tool 2016
Upgrade Guide 2019
Upgrading Payara Server 2019
Payara Server Upgrade Tool 2021
Backup and Restore Upgrade Method 2028
Domain and Node Directories Upgrade Method 2030
Payara Micro Documentation 2032
Payara Micro Documentation 2032

Documentation Conventions 2032
Docker Images 2033
Maven Support 2033
Maven Central Artifacts 2033
Run Payara Micro Using the Maven Plugin 2034
Run Payara Micro Using the Netbeans IDE 2034
Payara Micro Configuration and Management 2034
Micro Management 2034
Database Management 2083
Logging and Monitoring 2087
Logging 2087
Request Tracing in Payara Micro 2089
API 2093
Jcache in Payara Micro 2093
Payara Micro API 2094
Extensions 2107
JCA Support in Payara Micro 2107
Persistent EJB Timers 2108
Remote CDI Events in Payara Micro 2109
Running Callable Objects on Bootstrapped Instances 2110
Payara Micro Docker Image Overview 2111
Quick Start 2111
Exposed Ports 2111
Installation Details 2112
Application Deployment 2112
Configuration 2115
Payara Embedded Documentation 2118
Overview 2118

Payara Server Embedded Server Guide 2118
Introduction to Embedded Payara Server 2118
Embedded Payara Server File System 2118
Including the Payara Server Embedded Server API in Applications 2120
Using the EJB Embeddable API with Embedded Payara Server 2134
Changing Log Levels in Payara Server Embedded 2137
Default Jakarta Persistence Data Source for Embedded Payara Server 2138
Restrictions for Payara Server Embedded Web Profile 2139
Payara Server Embedded Use Cases 2140
JDK 17 Considerations 2141
See Also 2141
Application Development 2141
Overview 2141
Related Documentation 2141
Class Loaders 2142
Standard Classloading 2143
Enhanced Classloading 2152
Debugging Applications 2155
Debugging in Payara Server 2156
Debugging in Payara Micro 2160
Profiling Tools 2160
Securing Applications 2161
Security Goals 2161
Payara Server Specific Security Features 2162
Container Security 2162
Roles, Principals, and Principal to Role Mapping 2165
Realm Configuration 2168
The `server.policy` File 2174

Configuring Message Security for Web Services 2179
Programmatic Login Using the Programmaticlogin Class 2186
User Authentication for Single Sign-On 2190
Jakarta Security Support 2192
Jakarta Authentication Support 2199
Advanced Groups Configuration 2208
Advanced Principal Name Configuration 2211
Client Certificate Custom Validations 2213
JCE Provider Support 2216
Developing CDI Components 2219
Concurrent CDI Bean Loading 2219
Legacy Mode of Empty beans.xml 2220
Developing SOAP Web Services 2221
Deploying a Web Service 2221
The Web Service URI, WSDL File, and Test Page 2222
The Databinding Provider 2223
Additional Scanning Locations 2225
Configuring the Java Persistence Provider 2225
Eclipselink Overview 2226
Using Eclipselink in Payara Server 2227
Specifying the Database for an Application 2228
Specifying the Persistence Provider for an Application 2230
Primary Key Generation Defaults 2231
Automatic Schema Generation 2232
Restrictions and Optimizations 2236
Using the Hazelcast Eclipselink Cache Coordination Protocol 2241
Developing Web Applications 2242
Using Servlets 2242

Using Jakarta Server Pages 2247
Creating and Managing Sessions 2252
Using Comet 2259
Advanced Web Application Features 2263
Using Jakarta Faces Technology 2276
Introducing Jakarta Faces 2276
Configurable JSF Options 2277
Using an Alternative Jakarta Faces Implementation 2278
Using Jakarta MVC 2279
Introducing Jakarta MVC 2280
Dependencies Setup 2281
The Controller 2281
The View 2283
Models 2284
See Also 2287
Using Jakarta Enterprise Beans Technology 2287
Customized EJB-JAR Archive Names 2287
Value Added Features 2289
EJB Timer Service 2293
Using Session Beans 2299
Using Read-Only Beans 2309
Using Message-Driven Beans 2312
EJB Timers in Deployment Groups 2317
Tracing Remote Ejbs 2319
JDK 17+ Support Considerations 2322
Using Lite Remote EJB Technology 2322
Configuring HTTP(S) for Remote Ejbs 2324
Configuring the Thin Client Dependency 2324

Client-Side Adapter 2330
Known Limitations 2332
See Also 2332
Developing Java Clients 2332
Introducing the Application Client Container 2333
Developing Clients Using the ACC 2335
Developing Clients without the ACC 2359
Developing Connectors 2364
Connector Support in Payara Server 2365
Advanced Connector Configuration Options 2367
Inbound Communication Support 2374
Outbound Communication Support 2375
Configuring a Message Driven Bean to Use a Resource Adapter 2376
Developing Osgi-Enabled Jakarta EE Applications 2378
Overview of Osgi Application and Payara Server 2378
Developing Osgi Application Bundles for Payara Server 2380
Deploying Osgi Bundles in Payara Server 2390
Using the JDBC API for Database Access 2390
Statements 2391
SQL Trace Listeners 2394
Connections 2397
Connection Wrapping 2404
Allowing Non-Component Callers 2405
Using Application-Scoped Resources 2406
JDBC Development Guidelines 2406
Using the Transaction Service 2410
Handling Transactions with Databases 2410
Handling Transactions with Enterprise Beans 2414

Handling Transactions with Jakarta Messaging 2416
The Transaction Manager, the Transaction Synchronization Registry, and `UserTransaction` 2417
Using the Java Naming and Directory Interface 2418
Accessing the Naming Context 2418
Configuring Resources 2423
Using a Custom `jndi.properties` File 2428
Mapping References 2428
Using Jakarta Messaging 2430
Using Application-Scoped JMS Resources 2430
Load-Balanced Message Inflow 2430
Authentication with `ConnectionFactory` 2431
Delivering SOAP Messages Using the JMS API 2431
Using Jakarta Mail 2434
Introducing Jakarta Mail 2434
Creating a Jakarta Mail Session 2435
Jakarta Mail Session Properties 2435
Looking up a Jakarta Mail Session 2435
Sending and Reading Messages Using Jakarta Mail 2436
Using Application-Scoped Jakarta Mail Resources 2438
Using the Data Grid in Your Applications 2438
Accessing the Hazelcast Instance via JNDI 2438
Using Hazelcast for Web and EJB Container Persistence 2439
Using the Jcache API 2443
Accessing the Caching Provider and Cache Manager 2443
Creating a Cache Instance 2444
Using Jcache Annotations 2446
See Also 2446

Using Request Tracing in Applications . . . 2446
Terminology . . . 2446
Using the `@Traced` Annotation . . . 2447
Using the Opentracing `Tracer` Class . . . 2449
Disabling Automatic Tracing of JAX-RS Methods and Microprofile REST Client Calls . . . 2450
Tracing APIs Compatibility Matrix . . . 2451
Features Not Currently Supported . . . 2451
Public API . . . 2452
Public API . . . 2452
Use Cases . . . 2452
Maven Coordinates . . . 2452
Firing and Listening for Remote CDI Events . . . 2453
Requirements . . . 2453
Initializing the Event Bus . . . 2453
Firing Remote CDI Events . . . 2453
Observing Remote CDI Events . . . 2454
Additional Options . . . 2454
Dynamic Configuration . . . 2456
Clustered Singleton . . . 2457
How It Works . . . 2457
Examples . . . 2457
`@Clustered` and EJB Timers . . . 2457
`@Clustered` and `@Startup` / `@Initialized` / `@ApplicationScope` . . . 2457
CDI Considerations and Notes . . . 2458
Requirements and Assumption . . . 2458
Usage . . . 2459
Configuration . . . 2460

Distributed Locking 2460
Transactions 2461
OAuth2 Support 2461
Usage 2461
Configuration 2463
See Also 2464
Openid Connect Support 2464
Usage 2464
Configuration 2466
Expression Language Support 2472
Multi-Tenancy Support (Session-Scoped Configuration) 2473
Client Secret Aliasing 2481
Fetch Caller Data 2481
User Information from the ID Token 2483
Disable Scope Validation 2483
Bearer Authentication and Authorization 2484
Integration with Specific Providers 2485
See Also 2497
Rolespermitted Support 2497
Usage 2498
Configuration 2498
Dynamic Roles 2499
See Also 2501
Jakarta EE Security Extensions 2501
Custom Authentication Mechanisms 2501
Custom Identity Stores 2502
MicroProfile 2508
Eclipse Microprofile 2508

What Is Eclipse Microprofile? . . . 2508
Payara Support for Microprofile . . . 2509
What Does Each Specification Contain? . . . 2509
Config . . . 2509
Eclipse Microprofile Config API . . . 2509
Cloud . . . 2515
Directory Config Source . . . 2545
JDBC Config Source . . . 2549
LDAP Config Source . . . 2551
TOML Config Source . . . 2555
Eclipse Microprofile Fault Tolerance API . . . 2557
Background . . . 2557
Additions to the Fault Tolerance Spec . . . 2558
Fault Tolerance Configuration . . . 2559
Upgrading from Eclipse Microprofile 3.x to 4.x . . . 2560
Eclipse Microprofile Health Check API . . . 2561
Background . . . 2561
REST Endpoints . . . 2562
Health Check Configuration . . . 2562
Upgrading from Eclipse Microprofile 3.x to 4.x . . . 2564
Eclipse Microprofile JWT Authentication API . . . 2564
Background . . . 2564
Jakarta Security and JWT . . . 2565
Vending Tokens . . . 2565
Accepting Tokens . . . 2567
Activating JWT Authentication . . . 2572
Protecting Jakarta REST Services Endpoints . . . 2572
Accessing a Protected Endpoint . . . 2573

Switching off @Rolesallowed Support 2573
Metrics 2574
Metrics Configuration in Azul Payara 2574
REST Endpoint 2575
Custom Vendor Metrics 2576
Eclipse Microprofile Openapi API 2583
Background. 2583
REST Endpoint 2583
Openapi Configuration Sources 2584
Openapi Configuration 2590
Upgrading from Microprofile 3.x to 4.x 2591
Opentelemetry and Opentracing Support. 2594
Enabling Opentelemetry 2595
Configuring Opentelemetry. 2595
Manual Tracing. 2598
Using Additional Components 2599
Relation to Request Tracing Service 2601
Eclipse Microprofile Opentracing 2601
Background. 2602
Text and Binary Carriers Support 2602
JAX-WS Support. 2602
Opentracing Configuration 2602
Using Alternate Opentracing Implementations 2603
Eclipse Microprofile Rest Client API 2603
Background. 2603
Jakarta CDI 2603
Asynchronous Communication 2604
Providers 2604

SSL Configuration 2605
Proxy Configuration 2606
See Also 2608
Eclipse Microprofile Telemetry 2608
Ecosystem 2609
Azul Payara Ecosystem 2609
Maven Bill of Materials Artifact 2610
Maven Plugin 2616
Payara Micro Maven Archetype 2643
Payara Micro Gradle Plugin. 2644
Maven Regex Profile Activation Extension 2647
Payara Starter Documentation 2648
IDE Integration. 2672
Eclipse Plugin 2672
IntelliJ Plugin 2708
Apache NetBeans IDE 2732
VSCode Extension 2758
Hot Deploy and Auto Deploy 2780
Connector Suites. 2781
Arquillian Containers 2781
Cloud Connectors 2798
Security Connectors 2837
Upgrade Advisor Tool 2838
Usage 2838
Structure of the Advisor’s Summary 2839
Basic Showcase Example 2841
Miscellaneous 2842
JAX-RS Extension. 2842

Dependencies 2843
Payara Platform Dependencies 2843
Jakarta EE Specification Dependencies Mapping 2844
`activation` 2844
`annotation` 2844
`authentication` 2846
`authorization` 2846
`batch` 2847
`concurrency` 2848
`connectors` 2848
`context-dependency-injection` 2850
`data` 2856
`dependency-injection` 2858
`deployment` 2861
`ee-platform` 2862
`enterprise-beans` 2863
`expression-language` 2866
`faces` 2867
`interceptors` 2869
`json-binding` 2869
`json-processing` 2869
`mail` 2874
`management` 2876
`messaging` 2878
`pages` 2880
`persistence` 2881
`restful-webservices` 2883
`security` 2889

`servlet` ... 2893
`standard-tag-library` ... 2894
`transaction` ... 2895
`validation` ... 2896
`webservices` ... 2897
`websocket` ... 2908
`xml-binding` ... 2910
`xml-rpc` ... 2914
MicroProfile Specification Dependencies Mapping ... 2916
`Common` ... 2916
`Config` ... 2916
`Fault-Tolerance` ... 2917
`Health` ... 2917
`JWT-Authentication` ... 2918
`Metrics` ... 2919
`OpenAPI` ... 2919
`Rest-Client` ... 2920
`Telemetry` ... 2920
Payara Platform Internal Dependencies ... 2922
acc-config ... 2922
accessors-smart ... 2922
admin-cli ... 2923
admin-core ... 2923
admin-util ... 2923
amx-core ... 2923
amx-javaee ... 2923
ant ... 2924
aopalliance-repackaged ... 2924

api-exporter 2924
appclient-connector 2924
appclient-server-core 2924
asadmin-audit 2925
asadmin-recorder 2925
backup 2925
bcel 2925
class-model 2925
classgraph 2926
classmate 2926
cluster-admin 2926
cluster-common 2926
cluster-ssh 2927
common-util 2927
concurro 2927
config-api 2927
config-types 2927
console-cluster-plugin 2928
console-common 2928
console-common-full-plugin 2928
console-corba-plugin 2928
console-jca-plugin 2928
console-jdbc-plugin 2929
console-payara-enterprise-branding-plugin 2929
console-plugin-service 2929
console-reference-manual-plugin 2929
console-web-plugin 2929
container-common 2930

cpu-core-reporting 2930
crac 2930
discord-notifier-console-plugin 2930
discord-notifier-core 2931
docker 2931
dol 2931
environment-warning 2931
epicyro 2931
eventbus-notifier-console-plugin 2932
flashlight-extra-jdk-packages 2932
flashlight-framework 2932
gf-admingui-connector 2932
gf-client-module 2932
gf-restadmin-connector 2933
gf-web-connector 2933
glassfish 2933
glassfish-api 2933
glassfish-corba-csiv2-idl 2933
glassfish-corba-internal-api 2934
glassfish-corba-omgapi 2934
glassfish-corba-orb 2934
glassfish-ee-api 2934
glassfish-extra-jre-packages 2935
glassfish-grizzly-extra-all 2935
glassfish-javassist-packages 2935
glassfish-mbeanserver 2935
glassfish-naming 2935
glassfish-oracle-jdbc-driver-packages 2936

gmbal 2936
gmbal-api-only 2936
grizzly-npn-osgi 2936
ha-api. 2936
ha-file-store 2937
ha-hazelcast-store. 2937
hazelcast. 2937
hazelcast-bootstrap 2937
hazelcast-tpc-engine. 2937
healthcheck-checker 2938
healthcheck-core 2938
healthcheck-cpool 2938
healthcheck-metrics 2938
healthcheck-service-console-plugin 2939
healthcheck-stuck 2939
hikaricp 2939
hk2. 2939
hk2-api 2939
hk2-config. 2940
hk2-core 2940
hk2-extras. 2940
hk2-locator 2940
hk2-runlevel 2940
hk2-utils 2941
internal-api 2941
jakarta-ee11-shim 2941
jakarta-ee9-shim 2941
jakarta.mvc-api 2941

jansi-core 2942
javassist 2942
jboss-classfilewriter 2942
jboss-logging 2942
jdbc-admin 2943
jdbc-config 2943
jdbc-runtime 2943
jline 2943
jline-builtins 2943
jline-native 2944
jline-reader 2944
jline-remote-ssh 2944
jline-remote-telnet 2944
jline-style 2944
jline-terminal 2945
jline-terminal-ffm 2945
jline-terminal-jansi 2945
jline-terminal-jna 2945
jline-terminal-jni 2945
jsr107-repackaged 2946
kernel 2946
krazo-core 2946
launcher 2946
ldapbp-repackaged 2947
libpam4j 2947
libpam4j-repackaged 2947
logging 2947
microprofile-connector 2947

microprofile-console-plugin 2948
mimepull 2948
monitoring-console-api 2948
monitoring-console-core 2948
monitoring-console-process 2948
monitoring-core 2949
newrelic-notifier-console-plugin 2949
newrelic-notifier-core 2949
notification-core 2949
notification-eventbus-core 2949
nucleus-grizzly-all 2950
nucleus-resources 2950
orb-connector 2950
orb-enabler 2950
orb-iiop 2951
org.apache.felix.bundlerepository 2951
org.osgi.service.component 2951
org.osgi.util.function 2951
org.osgi.util.promise 2951
osgi-adapter 2952
osgi-cli-remote 2952
osgi-container 2952
osgi-resource-locator 2952
payara-api 2952
payara-console-extras 2953
payara-executor-service 2953
payara-insight-console-plugin 2953
payara-jsr107 2953

payara-micro-boot 2953
payara-micro-service 2954
payara-rest-endpoints 2954
pfl-basic 2954
pfl-basic-tools 2954
pfl-dynamic 2955
pfl-tf 2955
pfl-tf-tools 2955
phonehome-bootstrap 2955
realm-stores 2955
requesttracing-core 2956
rest-client 2956
rest-client-ssl 2956
rest-monitoring-plugin 2956
rest-monitoring-service 2956
rest-service 2957
rmic 2957
scattered-archive-api 2957
simple-glassfish-api 2957
slack-notifier-console-plugin 2957
slack-notifier-core 2958
snakeyaml 2958
snakeyaml-engine 2958
snmp-notifier-console-plugin 2958
snmp-notifier-core 2959
snmp4j 2959
snmp4j-repackaged 2959
ssl-impl 2959

stats77 2959
teams-notifier-console-plugin 2960
teams-notifier-core 2960
tiger-types 2960
trilead-ssh2 2960
trilead-ssh2-repackaged 2960
war-util 2961
wasp 2961
web-cli 2961
web-core 2961
web-embed-api 2961
web-glue 2962
web-gui-plugin-common 2962
web-ha 2962
web-naming 2962
web-sse 2963
woodstox-core 2963
xsdlib 2963
Release Notes **2963**
Release Notes 2963
All Release Notes 2963
Release Notes - Azul Payara 7.2.0 2964
Supported APIs and Applications 2964
Security Fixes 2964
Bug Fixes 2964
Improvements 2964
Component Upgrades 2964
Release Notes - Azul Payara 7.1.0 2965

Supported APIs and Applications 2965
Security Fixes 2965
Bug Fixes. 2965
Improvements 2966
Component Upgrades 2966
Release Notes - Payara Platform Enterprise 7.0.0 2968
Supported APIs and Applications 2968
Jakarta EE Certification. 2968
Jakarta EE Certification Summary. 2968
7.1.0. 2968
Platform TCK Results 2968
Payara Server 7.1.0 Jakarta EE 11 Platform Certification Summary. 2969
Test Results 2974
Web TCK Results. 2986
Payara Server 7.1.0 (Web Profile) Jakarta EE 11 Web Profile Certification Summary. 2986
Test Results 2989
7.0.0. 2995
Platform TCK Results 2995
Payara Server 7.0.0 Jakarta EE 11 Platform Certification Summary. 2995
Test Results 3000
Web TCK Results. 3012
Payara Server 7.0.0 (Web Profile) Jakarta EE 11 Web Profile Certification Summary. 3012
Test Results 3016
Eclipse MicroProfile Certification 3021
Eclipse Microprofile Certification Summary. 3021
Security 3022

Overview 3022
The CVE Program 3022
General Guidelines 3023
Vulnerabilities on Third-Party Components 3024
Reporting Security Vulnerabilities 3024
CVE ID Assignment 3024
CVE Record Disclosure 3025
CVE Record Rejection 3025
CVE Disputes 3025
Security Advisories 3026
Credits 3026
See Also 3026
Security Advisories 3027
Non-CVE Vulnerabilities 3027
Appendix **3028**
Schemas 3028
Payara Schemas 3028

General Info

Documentation Overview

This page provides some general information about the Azul Payara Edition.

Release Strategy

The Azul Payara Edition project is released each month. Each release incorporates fixes and enhancements from Platform engineering and Enterprise-only features.

Versioning Scheme

The general format of an Azul Payara version number is:

```
Major Version Number. Minor Version Number. Patch Version Number
```

Major Version numbers start from **6** (the current major version of Azul Payara) and only increment by 1 when a new major version is released. Fundamental changes will only be released in major Versions. There is currently no set cadence for this version change.

The change of a major version number is a product management and marketing decision.

TIP	Customers should expect changes to their applications and configuration scripts to move to a new major version.

Minor Version numbers start from **0** and increment by `1` as required. These version changes will introduce significant features (typically originating from the Product Management process or stabilised and adopted from Community releases).

NOTE	Features introduced during a minor version change must be backwards compatible with previous minor releases with no intervention required by an operator. Features introduced during a minor version change can only introduce breaking changes as a last resort and with product management

> approval and appropriate customer messaging.

The minor version number will reset to **0** on a major version release.

> **TIP** Customers should not expect to change their applications and configuration scripts to move to a minor version unless by choice to take advantage of newly introduced features.

Patch Version numbers will start from **0** and as a minimum increment as required. These version change will introduce bug fixes and patches. The Patch Version number will reset to **0** on a minor version release.

Customer specific versions used for private builds or hotfixes will add a 4th level number where the number reflects a meaningful name to the customer e.g. the number of the hotfix.

For example if a customer specific hotfix is provided for a bugfix identified with *1234* then the release number for that customer will be *6.1.0.1234*

> **TIP** Customers should not expect to change their applications and configuration scripts to move to a patch version.

Examples

> **NOTE** The examples shown below are used only to showcase possible future releases and are not be followed up as a release plan when planning updates on your organization.

- `6.0.0` = Major Version `6`, Minor Version `0`, Patch `0`
- `6.1.1` = Major Version `6`, Minor Version `1`, Patch `1`
- `6.1.2` = Major Version `6`, Minor Version `1`, Patch `2`
- `7.0.0` = Major Version `7`, Minor Version `0`, Patch `0`

Supported Platforms

Though it is always possible for new versions to be supported, this page details the JVMs currently supported by Azul Payara (both Payara Server and Payara Micro).

Supported Java Virtual Machines

Azul Payara currently runs on the following Java Virtual Machines:

Java Virtual Machine	Supported Version(s)
Azul Zulu JDK (Azul Core)	21 (21.30/21.0.1+), 25 (25.30/25.0.1+)
Azul Zing JDK (Azul Prime)	21 (21.30/21.0.1+), 25 (25.30/25.0.1+)
Oracle JDK	21 (21.0.1+), 25 (25.0.1+)
Amazon Corretto	21 (21.0.1+), 25 (25.0.1+)
Adopt Open JDK	21 (21.0.1+), 25 (25.0.1+)
Adopt Open JDK with Eclipse Open J9	21 (21.0.1+), 25 (25.0.1+)
Any other JVM based on OpenJDK	21.0.1+, 25 (25.0.1+)

Azul Payara runs on the `x64` and `arm64` variants of the above JVMs.

Supported Operating Systems

Azul Payara currently runs on the following Operating Systems in all the versions listed below and all their newer versions:

Family	Operating System	Minimum Version
Windows	Windows	7+
	Windows Server	2008+

Family	Operating System	Minimum Version
Linux	Ubuntu (Trusty Tahr)	14.04 LTS+
	Debian (Wheezy)	7+
	CentOS	6+
	RHEL	5+
	OpenSUSE	42.2+
	SUSE	11.4, 12.2+, 15.0+
	Amazon Linux	All versions
Other	MacOS (OSX) (Yosemite)	10.10.5+
	IBM AIX	7.1+

In general, any Azul Payara distribution runs on any system supported by at least one of the JVMs it runs on.

Third Party Collaborations

Azul Payara integrates with various other services that complement your Java deployment experience on the platform.

Azul

Payara's partnership with Azul brings Azul Core and Azul Prime to your application deployment.

Azul Core

Azul Core is designed for enterprise, distribution (ISV), and embedded (OEM) use cases. Azul Core provides certified OpenJDK builds, Azul Zulu Builds of OpenJDK (Azul Zulu), with tight security and the cost efficiencies you need to run today's business–critical, Java-based services. These builds are free to use and a drop-in replacement for any other OpenJDK distribution.

Key advantages include:

- **High Trust**: Chosen by leading global brands, Azul Core offers engineering support and 100% open-source builds that are Java SE compliant and TCK verified.
- **High Value**: Azul Core is a one-to-one replacement for Oracle Java SE with the most architectures, package types, and configurations options compared to other OpenJDK providers.
- **High Security**: Azul Core delivers quarterly stabilized security updates for rapid deployment into production, backed by an industry-leading support SLA.

Read more about Azul Core and Azul Zulu JDK here: Azul Core Documentation

Azul Prime

Azul Prime is a high-performance Java platform that delivers up to 37% faster performance compared to other OpenJDK distributions. At its core is Azul Zing, an enhanced build of OpenJDK that serves as the actual JDK running your applications. The platform requires no code changes or recompilation, making integration straightforward for existing Java applications.

Key advantages include:

- **Performance Improvements**: Azul Zing, the enhanced JDK that runs your applications, achieves improved responsiveness and consistency while maintaining performance SLAs, even as loads increase. It provides superior application performance, responsiveness, and efficiency compared to standard OpenJDK distributions.
- **Cost Reduction**: Organizations can reduce cloud compute costs by 20% or more through Azul Zing's more efficient CPU utilization and faster Java code execution, requiring fewer compute resources for the same workload.
- **Advanced Garbage Collection**: Azul Zing includes the C4 Pauseless Garbage Collector, which ensures consistent performance under heavy loads and enables 2-5X more transaction processing without pauses, eliminating the "stop-the-world" pauses that limit scalability in traditional JVMs.

- **Rapid Application Startup**: Azul Zing's ReadyNow technology speeds up application startup and time to peak performance by reusing profiling data, solving traditional Java warm-up issues and allowing applications to start fast and stay fast.
- **Optimizer Hub**: This optional Azul Prime component provides centralized JIT compilation and intelligent curation of optimization profiles, enabling fleet-wide collaboration where multiple Azul Zing JVMs share optimizations and learnings for improved cloud scale, elasticity, and cost efficiencies.

Azul Prime requires opt-in during contract negotiations with sales. Sales will acquire a subscription based on vCPU cores, with annual billing directly from Azul. Customers receive an installation wizard with the JVM enabled. For post-installation enablement, contact sales - the service team will provide two commands to activate it on your existing installation.

Read more about Azul Prime and Azul Zing JDK here: Azul Prime Documentation

Technical Documentation

Payara Server Documentation

Payara Server Documentation

Comprehensive documentation of how to operate and manage a Payara Server installation composed of the following subsections:

- **General Administration**: Details the basics of how to manage the configuration and runtime settings of Payara Server.
- **Deployment Planning**: Contains instructions and recommendations on how to plan an application deployment in Payara Server.
- **Application Deployment**: Details how application deployments work in Payara Server.
- **High Availability**: Details how to configure and use the high-availability features of Payara Server on distributed environments.
- **Security Guide**: A comprehensive guide that offers basic recommendations and

advice on how to secure a Payara Server installation.

- **Command Reference**: A complete reference manual of alf available Asadmin CLI commands.
- **Extensions**: Documents the features of all available extensions for Payara Server and how to use them.
- **Upgrade Guide**: A guide with recommendations and strategies for upgrading Payara Server across versions.

Documentation Conventions

The following conventions are applied through the pages in this section to improve readability and familiarity with how Payara Server operates.

Symbol Conventions

The following table explains symbols that might be used in this section.

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference or a placeholder reference for a default path.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
-	Joins simultaneous multiple keystrokes.	`Control-A`	Press the `Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	`Ctrl+A+N`	Press the Control key, release it, and then press the subsequent keys.

Symbol	Description	Example	Meaning
□	Indicates menu item selection in a graphical user interface.	File □ New □ Templates	From the *File* menu, choose New. From the *New* submenu, choose *Templates*.

Default Paths and File Names

The following table describes the default paths and file names that are used in this book.

Placeholder	Description	Default Value

`as-install`	Represents the base installation directory for Payara Server.
	
	In configuration files, as-install is represented as follows:
	
	`${com.sun.aas.installRoot}`
		Installations on a Linux operating system, and macOS operating system:
		
		`{USER_HOME}/payara7/glassfish`
		
		Installations on a Windows operating system:
		
		`{SystemDrive}:\payara7\glassfish`

`as-install-parent`	Represents the parent of the base installation directory for Payara Server.	Installations on a Linux operating system, and Mac operating system:
		
		`{USER_HOME}/payara7`
		
		Installations on a Windows operating system:
		
		`{SystemDrive}\payara7`

`domain-root-dir`	Represents the directory in which a domain is created by default.	`as-install/domains/`
`domain-dir`	Represents the directory in which a domain's configuration is stored. In configuration files, `domain-dir` is represented as follows: `${com.sun.aas.instanceRoot}`	`domain-root-dir/<domain-name>`

Placeholder	Description	Default Value
`node-dir`	Represents the directory for a configured node.	`as-install/glassfish/nodes/<node-name>`
`instance-dir`	Represents the directory for a server instance.	`node-dir/<instance-name>`

Docker Images

Payara Services Limited provides official Payara Server Docker Images which are updated every release.

These images are hosted on Docker hub in the following locations:

- Payara Server Full Profile Docker Image
- Payara Server Web Profile Docker Image

> **TIP** Payara Enterprise customers have access to a private Docker repository containing Payara Enterprise Edition equivalent images, which are also released and updated every month.

General Administration

Overview of Payara Server Administration

Payara Server provides a runtime for developing and deploying Jakarta EE Platform and Eclipse MicroProfile applications.

As an administrator of Payara Server, your main responsibilities are to establish a secure Payara Server environment and to oversee the services, resources, and users that participate in that environment.

Your key tasks include configuring resources and services, managing Payara Server at runtime, and fixing problems that are associated with the server's operation.

Default Settings and Locations

After installation, you might need to perform some immediate configuration tasks to

make your installation function as intended.

If configuration defaults have been accepted, some features are enabled and some not. For an overview of initial configuration tasks for Payara Server services and resources, see initial-configuration-tasks.

In addition, you might want to reset default passwords, change names or locations of files, and so on.

The following tables list the default administration values.

NOTE The default administrator login is `admin`, with no password, which means that no login is required.

Default Administration Values

Item	Default
Domain Name	`domain1`
Master Password	`changeit`
Administration User	`admin`
Administration Server Port	`4848`
HTTP Port	`8080`
HTTPS Port	`8181`
Pure JMX Clients Port	`8686`
Message Queue Port	`7676`
IIOP Port	`3700`
IIOP/SSL Port	`3820`
IIOP/SSL Port With Mutual Authentication	`3920`

Table 1. Default Locations

Item	Default
Command-line Utility (`asadmin`)	`as-install/bin`
Configuration Files	`domain-dir/config`
Log Files	`domain-dir/logs`

Configuration Tasks

Some configuration tasks must be performed directly after installation for your Payara Server environment to work as intended. For example, if you are using a database with Payara Server, you need to set up database connectivity right away.

Some configuration situations are ongoing and will require you to make changes many times during the life of your installation. You can use either the Administration Console or the `asadmin` utility to modify the configuration. Changes are automatically applied to the appropriate configuration file.

Initial Configuration Tasks

This section maps the common configuration tasks to the command-line procedures in this guide. In some situations, the resource or service is automatically enabled and your configuration tasks involve adjusting or changing the default settings to suit your specific needs.

The following resources and services frequently require configuration immediately after installation:

- **System Properties**: See administering-system-properties.
- **Domains**: The initial `domain1` is created during installation. Additional configuration tasks might include such tasks as configuring additional domains or setting up automatic restart. See administering-domains.
- **JVM**: The initial tasks for configuring the JVM include creating JVM options.
- **Logging**: By default, logging is enabled, so basic logging works without additional configuration. However, you might want to change log levels, property values, or the location of log files. See administering-the-logging-service.

- **Monitoring**: By default, the monitoring service is enabled. However, monitoring for the individual modules is not enabled, so your first monitoring task is to enable monitoring for the modules that you want to monitor. See administering-the-monitoring-service.
- **Security**:
  - System Security. Initial configuration tasks might include setting up passwords and certificates. See "administering-system-security" in the Payara Server Security Guide.
  - User Security. Initial configuration tasks might include creating authentication realms and file users. See "administering-user-security" in the Payara Server Security Guide.
  - Message Security. Initial configuration tasks might include configuring a Java Cryptography Extension (JCE) provider, enabling default and non-default security providers, and configuring message protection policies. See "administering-message-security" in the Payara Server Security Guide.
- **Database Connectivity**: The initial tasks involved in configuring Payara Server to connect to an external database include creating a Java Database Connectivity (JDBC) connection pool, creating a JDBC resource, and integrating a JDBC driver. See administering-database-connectivity.
- **EIS Connectivity**: The initial tasks involved in configuring Payara Server to connect to an enterprise information system (EIS) include creating a connector connection pool, creating a connector resource, editing a resource adapter configuration, creating a connector security map, creating a connector work security map, and creating an administered object (if needed). See administering-eis-connectivity.
- **Internet Connectivity**: The initial tasks involved in making deployed web applications accessible by internet clients include creating HTTP network listeners and virtual servers, and configuring the HTTP listeners for SSL (if needed). See administering-http-connectivity.

- **Object Request Broker (ORB)**: An initial configuration task might involve creating an IIOP listener. See administering-the-object-request-broker-orb.
- **JavaMail Service**: An initial configuration task might involve creating a JavaMail resource. See administering-the-javamail-service.
- **Java Message Service (JMS)**: Initial configuration tasks might include creating a physical destination, creating connection factories or destination resources, creating a JMS host (if the default JMS host is not adequate), adjusting connection pool settings (if needed), and configuring resource adapters for JMS. See administering-the-java-message-service-jms.
- **JNDI Service**: An initial configuration task might involve creating a JNDI resource. See administering-the-java-naming-and-directory-interface-jndi-service.

Information and instructions for accomplishing the tasks by using the Administration Console are contained in the Administration Console online help.

How Dotted Names Work for Configuration

After the initial configuration is working, you will continue to manage ongoing configuration for the life of your Payara Server installation.

You might need to adjust resources to improve productivity, or issues might arise that require settings to be modified or defaults to be reset. In some situations, an `asadmin` subcommand is provided for updating, such as the `update-connector-work-security-map` subcommand. However, most updating is done by using the `list`, `get`, and `set` subcommands with dotted names. For detailed information about dotted names, see the "`dotted-names`(5ASC)" help page.

NOTE	Dotted names also apply to monitoring, but the method is slightly different. For information on using dotted names for monitoring, see how-the-monitoring-tree-structure-works.

The general process for working with configuration changes on the command line is as follows:

- List the modules for the component of interest.

  The following single mode example uses the `|` (pipe) character and the `grep` command to narrow the search:

```
asadmin list "*" | grep http | grep listener
```

  Information similar to the following is returned:

```
configs.config.server-config.network-config.network-listeners.network-
listener.http-listener-1
configs.config.server-config.network-config.network-listeners.network-
listener.http-listener-2
configs.config.server-config.network-config.protocols.protocol.admin-
listener.http
configs.config.server-config.network-config.protocols.protocol.admin-
listener.http.file-cache
configs.config.server-config.network-config.protocols.protocol.http-listener-1
configs.config.server-config.network-config.protocols.protocol.http-listener-
1.http
configs.config.server-config.network-config.protocols.protocol.http-listener-
1.http.file-cache
configs.config.server-config.network-config.protocols.protocol.http-listener-2
configs.config.server-config.network-config.protocols.protocol.http-listener-
2.http
configs.config.server-config.network-config.protocols.protocol.http-listener-
2.http.file-cache
configs.config.server-config.network-config.protocols.protocol.http-listener-
2.ssl
```

- Get the attributes that apply to the module you are interested in.

  The following multimode example gets the attributes and values for `http-listener-1`:

```
asadmin> get server-config.network-config.network-listeners.network-
listener.http-listener-1.*
```

  Information similar to the following is returned:

```
server.http-service.http-listener.http-listener-1.acceptor-threads = 1
server.http-service.http-listener.http-listener-1.address = 0.0.0.0
server.http-service.http-listener.http-listener-1.blocking-enabled = false
server.http-service.http-listener.http-listener-1.default-virtual-server =
server
server.http-service.http-listener.http-listener-1.enabled = true
server.http-service.http-listener.http-listener-1.external-port =
server.http-service.http-listener.http-listener-1.family = inet
```

```
server.http-service.http-listener.http-listener-1.id = http-listener-1
server.http-service.http-listener.http-listener-1.port = 8080
server.http-service.http-listener.http-listener-1.redirect-port =
server.http-service.http-listener.http-listener-1.security-enabled = false
server.http-service.http-listener.http-listener-1.server-name =
```

- Modify an attribute by using the `set` subcommand.

  This example sets the `security-enabled` attribute of `http-listener-1` to true:

```
asadmin> set server.http-service.http-listener.http-listener-1.security-enabled
= true
```

Configuration Files

The bulk of the configuration information about Payara Server resources, applications, and instances is stored in the `domain.xml` configuration file. This file is the central repository for a given administrative domain and contains an XML representation of the Payara Server domain model. The default location for the `domain.xml` file is `domain-dir/config`.

IMPORTANT	Payara Server maintains a backup of the `domain.xml` file that is named `domain.xml.bak`. The purpose of this file is solely to enable Payara Server to start a domain if the `domain.xml` file cannot be read. Do not modify or delete the `domain.xml.bak` file and do not use this file for any other purpose.

The `logging.properties` file is used to configure logging levels for individual modules. The default `logging.properties` file is located in the same directory as the `domain.xml` file. For further information on the `logging.properties` file, see logging-properties.

The `asenv.conf` file is located in the `as-install/config` directory. Its purpose is to store the Payara Server environment variables, such as the installation location of the database, Message Queue, and so on.

CAUTION	Changes are automatically applied to the appropriate configuration

> file. Do not edit the configuration files directly. Manual editing is prone to error and can have unexpected results.

Impact of Configuration Changes

Some configuration changes require that you restart the DAS or Payara Server instances for the changes to take effect. Other changes are applied dynamically without requiring that the DAS or instances be restarted. The procedures in this guide indicate when a restart is required. Payara Server enables you to determine whether the DAS or an instance must be restarted to apply configuration changes.

Some changes to resources or connection pools affect the applications that use the resources or connection pools. These changes do not require restart. However, any applications that use the resources or connection pools must be disabled and re-enabled or redeployed for the change to take effect.

To Determine Whether the DAS or an Instance Requires Restart

1. Ensure that the DAS is running.
   To obtain information about the DAS or an instance, a running server is required.
2. Do one of the following:
   - To determine if the DAS requires restart, list the domains in your Payara Server installation.
     Use the "`list-domains`" subcommand for this purpose.

     ```
     asadmin> list-domains [--domaindir domain-root-dir]
     ```

     The domain-root-dir is the directory that contains the directories in which individual domains' configuration is stored. The default is `as-install/domains`, where `as-install` is the base installation directory of the Payara Server software.

     If the DAS requires restart, a statement that restart is required is displayed.

   - To determine if an instance requires restart, list information about the instance.

Use the "`list-instances`" subcommand for this purpose:

```
asadmin> list-instances instance-name
```

The instance-name is the name of the instance for which you are listing information.

If the instance requires restart, one of the following pieces of information is displayed: a statement that restart is required, or a list of configuration changes that are not yet applied to the instance.

Example 1-1 Determine Whether the DAS or an Instance Requires Restart

This example determines that the DAS for the domain `domain1` requires restart to apply configuration changes.

```
asadmin> list-domains
domain1 running, restart required to apply configuration changes
Command list-domains executed successfully.
```

Example 1-2 Determine Whether the DAS or an Instance Requires Restart

This example determines that the instance `pmd-i1` requires restart to apply configuration changes.

```
asadmin> list-instances pmd-i1
pmd-i1   running;  requires restart
Command list-instances executed successfully.
```

- "`list-domains`"
- "`list-instances`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line.

- `asadmin help list-domains`
- `asadmin help list-instances`

Configuration Changes that Require Restart

The following configuration changes require restart for the changes to take effect:

- Changing JVM options
- Changing port numbers
- Changing log handler elements
- Configuring certificates
- Managing HTTP, JMS, IIOP, JNDI services
- Enabling or disabling secure administration as explained in "running-secure-admin" in the Payara Server Security Guide

Dynamic Configuration Changes

With dynamic configuration, changes take effect while the DAS or instance is running. The following configuration changes do not require restart:

- Adding or deleting add-on components
- Adding or removing JDBC, JMS, and connector resources and pools (Exception: Some connection pool properties affect applications.)
- Changing a system property that is not referenced by a JVM option or a port
- Adding file realm users
- Changing logging levels
- Enabling and disabling monitoring
- Changing monitoring levels for modules
- Enabling and disabling resources and applications
- Deploying, un-deploying, and redeploying applications

Changes that Affect Applications

Some changes to resources or connection pools affect the applications that use the resources or connection pools. These changes do not require restart. However, any

applications that use the resources or connection pools must be disabled and re-enabled or redeployed for the change to take effect.

NOTE

If you do not know which applications use the changed resources or connection pools, you can apply these changes by restarting the clusters or Payara Server instances to which applications are deployed. However, to minimize the disruption to the services that your applications provide, avoid restarting clusters or instances to apply these changes if possible.

The following changes affect applications:

- Creating or deleting resources (Exception: Changes to some JDBC, JMS, or connector resources do not affect applications.)
- Modifying the following JDBC connection pool properties:
  - `datasource-classname`
  - `associate-with-thread`
  - `lazy-connection-association`
  - `lazy-connection-enlistment`
  - JDBC driver vendor-specific properties
- Modifying the following connector connection pool properties:
  - `resource-adapter-name`
  - `connection-definition-name`
  - `transaction-support`
  - `associate-with-thread`
  - `lazy-connection-association`
  - `lazy-connection-enlistment`
  - Vendor-specific properties

Administration Tools

For the most part, you can perform the same tasks by using either the graphical Administration Console or the `asadmin` command-line utility, however, there are exceptions.

Administration Console

The Administration Console is a browser-based utility that features an easy-to-navigate graphical interface that includes extensive online help for the administrative tasks.

To use the Administration Console, the domain administration server (DAS) must be running. Each domain has its own DAS, which has a unique port number. When Payara Server was installed, you chose a port number for the DAS, or used the default port of 4848. You also specified a username and password if you did not accept the default login (`admin` with no password).

When specifying the URL for the Administration Console, use the port number for the domain to be administered. The format for starting the Administration Console in a web browser is `http://hostname:port`. For example:

```
https://admin.payara.fish:4848
```

If the Administration Console is running on the host where Payara Server was installed, specify `localhost` for the host name. For example:

```
http://localhost:4848
```

If the Administration Console is run on a host different from the host where Payara Server was installed, a secure connection (`https` instead of `http`) is used. Some browsers do not display pages on secure connections by default and must be configured to permit secure protocols (SSL and TLS).

You can display the help material for a page in the Administration Console by clicking the Help button on the page. The initial help page describes the functions and fields of the page itself.

If you try to use the Administration Console from a system through a proxy server on another system back to the original system, while using the system's full host name (instead of `localhost` or `127.0.0.1`) you are denied access because the request is treated as a remote request, which requires that the secure administration feature (secure admin) to be enabled.

To avoid this situation, do one of the following:

- Do not use a proxy server.
- Use `localhost` or `127.0.0.1` as the host name.
- Enable secure admin so that what Payara Server interprets as a remote request is accepted as such.

To enable secure admin, see ""Managing Administrative Security"" in the Payara Server Security Guide.

`asadmin` Utility

The `asadmin` utility is a command-line tool that runs subcommands for identifying the operation or task that you want to perform. You can run `asadmin` subcommands either from a command prompt or from a script.Running `asadmin` subcommands from a script is helpful for automating repetitive tasks. Basic information about how the `asadmin` utility works can be found in the asadmin-1m help page. For instructions on using the `asadmin` utility, see using-the-asadmin-utility.

To issue an `asadmin` subcommand in the standard command shell (single mode), go to the `as-install/bin` directory and type the `asadmin` command followed by a subcommand. For example:

```
asadmin list-jdbc-resources
```

You can invoke multiple command mode (multimode) by typing `asadmin` at the command prompt, after which the `asadmin>` prompt is presented. The `asadmin` utility continues to accept subcommands until you exit multimode and return to the standard

command shell. For example:

```
asadmin> list-jdbc-resources
```

You can display a help page for any `asadmin` subcommand by typing `help` before the subcommand name. For example:

```
asadmin> help restart-domain
```

or

```
asadmin help restart-domain
```

A collection of the `asadmin` help pages is available in HTML and PDF format in the "Payara Server Reference Manual".

REST Interfaces

Payara Server provides representational state transfer (REST) interfaces to enable you to access monitoring and configuration data for Payara Server, including data that is provided by newly installed add-on components. For more information, see using-rest-interfaces-to-administer-payara-server.

Osgi Module Management Subsystem

The OSGi module management subsystem that is provided with Payara Server is the Apache Felix OSGi framework .

To administer this framework, use either the Apache Felix Gogo remote shell. This shell is provided with Payara Server. The shell uses the Felix Gogo shell-service to interact with the OSGi module management subsystem.

This tool allows you to perform administrative tasks on OSGi bundles such as:

- Browsing installed OSGi bundles
- Viewing the headers of installed OSGi bundles

- Installing OSGi bundles
- Controlling the life cycle of installed bundles

To Enable the Apache Felix Gogo Remote Shell

By default, the Apache Felix Gogo remote shell in Payara Server is disabled. Before using the shell to administer OSGi bundles in Payara Server, you must enable the shell.

Enabling the Apache Felix Gogo remote shell in Payara Server involves changing the value of the property `glassfish.osgi.start.level.final`. This property controls whether the OSGi start level service enables the shell when the DAS or a Payara Server instance is started.

1. Ensure that the DAS is running.
2. Change the value of the `glassfish.osgi.start.level.final` property from `2` to `3`.

   If the domain includes clustered or standalone instances on remote hosts, perform this step on each remote host. You can change this value either by creating a Java system property or by editing a file.

   To change this value by creating a Java system property, create the system property `glassfish.osgi.start.level.final` with a value of `3`:

   ```
   asadmin> create-jvm-options --target target -Dglassfish.osgi.start.level.final=3
   ```

   To change this value by editing a file, edit the plain-text file `as-install/config/osgi.properties` to change the value of the `glassfish.osgi.start.level.final` property from `2` to `3`.
3. Restart the DAS.

For instructions, see to-restart-a-domain.

To Run Apache Felix Gogo Remote Shell Commands

The Apache Felix Gogo remote shell is integrated with the Payara Server `asadmin` command line utility. You can use the `asadmin` subcommands `osgi` and `osgi-shell` to access the remote shell and run OSGi shell commands.

To Run Remote Shell Commands Using the `osgi` Subcommand

The `osgi` subcommand delegates the command line to the Apache Felix Gogo remote shell for the execution of OSGi shell commands. Commands are executed by the remote shell and results are returned by the `asadmin` utility. The `osgi` subcommand is supported in remote mode only.

1. Ensure that the server is running. Remote commands require a running server.
2. Access the remote shell by using the "`osgi`" subcommand. For the full syntax and options for this subcommand, see `osgi`.

To Run Remote Shell Commands Using the `osgi-shell` Subcommand

The `osgi-shell` subcommand provides interactive access to the Apache Felix Gogo remote shell for the execution of OSGi shell commands. OSGi shell commands are executed on the server and results are printed on the client. You can run multiple commands from a file or run commands interactively. The `osgi-shell` subcommand is supported in local mode only. Unlike other local subcommands, however, the DAS and the server instance whose shell is being accessed must be running.

1. Ensure that the server is running.
2. Access the remote shell by using the redeploy subcommand. For the full syntax and options for this subcommand, see `osgi-shell`.

Example 1-3 to Run Remote Shell Commands Using the Osgi-Shell Subcommand

This example lists Apache Felix Gogo remote shell commands. Some lines of output are omitted from this example for readability.

```
asadmin> osgi help

felix:bundlelevel
felix:cd
```

```
felix:frameworklevel
gogo:cat
gogo:each
gogo:echo

asadmin> osgi-shell
Use "exit" to exit and "help" for online help.

gogo$ help
felix:bundlelevel
felix:cd
felix:frameworklevel
gogo:cat
gogo:each
gogo:echo
```

Example 1-4 to Run Remote Shell Commands Using the Osgi-Shell Subcommand 2

This example runs the Felix Remote Shell Command `lb` without any arguments to list all installed OSGi bundles. Some lines of output are omitted from this example for readability.

```
asadmin> osgi lb
START LEVEL 2
ID|State       |Level|Name
 0|Active      |    0|System Bundle
 1|Active      |    1|Metro Web Services API OSGi Bundle
 2|Active      |    1|jakarta.annotation API
Command osgi executed successfully.

asadmin> osgi-shell
Use "exit" to exit and "help" for online help.
gogo$ lb
START LEVEL 2
ID|State       |Level|Name
 0|Active      |    0|System Bundle
 1|Active      |    1|Metro Web Services API OSGi Bundle
 2|Active      |    1|jakarta.annotation API
gogo$
```

Example 1-5 to Run Remote Shell Commands Using the Osgi-Shell Subcommand 3

This example runs the Felix Remote Shell Command `inspect` with the `service` option and the `capability` option to determine the services that OSGi bundle `251` provides. Some lines of output are omitted from this example for readability.

```
asadmin> osgi inspect service capability 251
Payara EJB Container for OSGi Enabled EJB Applications (251) provides services:
--------------------------------------------------------------------------------
objectClass = org.glassfish.osgijavaeebase.Extender
service.id = 68
```

```
[source,text]
-----
objectClass = org.glassfish.osgijavaeebase.OSGiDeployer
service.id = 69
service.ranking = -2147483648
Command osgi executed successfully.
...
asadmin> osgi -shell
Use "exit" to exit and "help" for online help.
gogo$ inspect service capability 251
Payara EJB Container for OSGi Enabled EJB Applications (251) provides services:
------------------------------------------------------------------------
objectClass = org.glassfish.osgijavaeebase.Extender
service.id = 68
...
gogo$
[source,text]
```

Java Monitoring and Management Console (JConsole)

Java SE provides tools to connect to an MBean server and view the MBeans that are registered with the server. JConsole is one such popular JMX Connector Client and is available as part of the standard Java SE distribution.

For instructions on integrating JConsole in the Payara Server environment, see configuring-jconsole-to-view-payara-server-monitoring-data.

Instructions for Administering Payara Server

Information and instructions on performing most of the administration tasks from the command line are provided in this document and in the `asadmin` utility help pages. For instructions on accessing `asadmin` online help, see to-display-help-information-for-the-asadmin-utility-or-a-subcommand.

Phone Home

All Azul Payara distributions report back a limited amount of information about their specific configuration at launch time. No sensitive data about your machine or specific usage of Payara Server is sent to 0s or any member of the Payara Services team.

Gathered phone home data is sent via a `GET` request to http://www.payara.fish/phonehome

The following is the list of information attributes gathered by the Phone Home service:

Attribute	Example
Payara Server Version	7.2.0 #badassfish
Java Virtual Machine Version	1.8.0_121
Domain Uptime	36500
Number of nodes in the domain	1
Number of instances in the domain	2
Randomly generated UUID	eaa6fe2c-2388-4aa7-fdcc-288c7b318ddfeaa6ef23-2388-4aa7-8dcc-288c7b318ddf

This is the only data which the phone home service gathers from your system.

NOTE The UUID is specific to the DAS and is randomly generated on startup, so it is not machine specific.

What is the data used for?

The phone home data is presently used to gather usage statistics about the corresponding distribution, such as how long users typically run instances, and what versions they use.

Previously this could only be done at an extremely simple level from downloads over time, giving us no idea how Payara Server is used beyond specific interactions with contributors and customers which may not accurately represent the state of our user base.

The phone home service was designed as a non-intrusive survey tool which would only gather data specific to Payara and has no contact with any running applications, or indeed anything on your machine other than the mentioned attributes.

Disabling and Enabling Phone Home

The phone home service is enabled by default when starting a Payara Server. The phone home service can be easily disabled in the following ways.

Enable the Phone Home service

```
asadmin> enable-phone-home
[source,text]
```

See enable-phone-home

Disable the Phone Home service

Using Asadmin

Run this asadmin command and then restart the server to disable PhoneHome

```
asadmin> disable-phone-home
[source,text]
```

See disable-phone-home

Disable by Removing The Service Module

- Navigate to the `${PAYARA_INSTALL_DIR}/glassfish/modules` directory
- Delete the `phonehome-bootstrap.jar` module.

List the current status of the Phone Home service.

```
asadmin> list-phone-home
[source,text]
```

See list-phone-home

See Also

- "Disable PhoneHome in Payara Micro"
- Phone Home Source Code

General Runtime Administration

This chapter provides instructions for performing general administration tasks in the Payara Server environment by using the `asadmin` command-line utility.

Using the `asadmin` Utility

Use the `asadmin` utility to perform administrative tasks for Payara Server from the

command line or from a script. You can use this utility instead of the Administration Console interface.

Path to the `asadmin` Utility

The `asadmin` utility is located in the `as-install/bin` directory. To run the `asadmin` utility without specifying the path, ensure that this directory is in your path.

`asadmin` Utility Syntax

The syntax for running the `asadmin` utility is as follows:

```
asadmin [asadmin-util-options] [subcommand [subcommand-options] [operands]]
```

The replaceable items in this syntax are described in the subsections that follow.

Subcommands of the `asadmin` Utility

The subcommand identifies the operation or task that you are performing. Subcommands are case-sensitive. Each subcommand is either a local subcommand or a remote subcommand.

- A local subcommand can be run without a running domain administration server (DAS). However, to run the subcommand and have access to the installation directory and the domain directory, the user must be logged in to the machine that hosts the domain.
- A remote subcommand is always run by connecting to a DAS and running the subcommand there. A running DAS is required.

`Asadmin` Utility Options and Subcommand Options

Options control the behavior of the `asadmin` utility and its subcommands. Options are case-sensitive.

The `asadmin` utility has the following types of options:

`asadmin` **utility options**: These options control the behavior of the `asadmin` utility, not the subcommand.

+ The `asadmin` utility options may precede or follow the subcommand, but `asadmin` utility options after the subcommand are deprecated.

+ All `asadmin` utility options must either precede or follow the subcommand. If `asadmin` utility options are specified both before and after the subcommand, an error occurs.

Subcommand Options: These options control the behavior of the subcommand, not the `asadmin` utility. Subcommand options must follow the subcommand.

A subcommand option may have the same name as an `asadmin` utility option, but the effects of the two options are different.

Options have a long form and a short form.

- The short form of an option has a single dash (`-`) followed by a single character.
- The long form of an option has two dashes (`--`) followed by an option word.

For example, the short form and the long form of the option for specifying terse output are as follows:

- Short form: `-t`
- Long form: `--terse`

> **TIP** Most options require argument values, except `Boolean` options, which toggle to enable or disable a feature.

Operands of `asadmin` Utility Subcommands

Operands specify the items on which the subcommand is to act. Operands must follow the argument values of subcommand options, and are set off by a space, a tab, or double dashes (`--`). The `asadmin` utility treats anything that follows the subcommand options and their values as an operand.

To Run an `asadmin` Utility Subcommand in Single Mode

In single mode, you must type a separate `asadmin` command for each subcommand that you want to use. After the subcommand has run, you are returned to the operating system's command shell. Any `asadmin` utility options must be specified in each separate `asadmin` command that you run.

If you require the same `asadmin` utility options for multiple subcommands, use the `asadmin` utility in multimode. For more information, see to-start-a-multimode-session.

1. In the operating system's command shell, run the `asadmin` utility, specifying the subcommand.
2. If necessary, also specify any required `asadmin` utility options, subcommand options, and operands.

Example 2-1 Running an Asadmin Utility Subcommand in Single Mode

```
asadmin list-applications
hello <web>
Command list-applications executed successfully.
```

This example runs the "`list-applications`" subcommand in single mode. In this example, the default values for all options are used.

The example shows that the application `hello` is deployed on the local host.

Example 2-2 Specifying an Asadmin Utility Option with a Subcommand in Single Mode

```
asadmin --host srvr1.example.com list-applications
basic-ezcomp <web>
scrumtoys <web>
ejb31-war <ejb, web>
automatic-timer-ejb <ejb>
Command list-applications executed successfully.
```

This example specifies the `--host asadmin` utility option with the `list-applications` subcommand in single mode. In this example, the DAS is running on the host `srvr1.example.com`.

The example shows that the applications `basic-ezcomp`, `scrumtoys`, `ejb31-war`,

and `automatic-timer-ejb` are deployed on the host `srvr1.example.com`.

Example 2-3 Specifying an `asadmin` Utility Option and a Subcommand Option in Single Mode

```
asadmin --host srvr1.example.com list-applications --type web
basic-ezcomp <web>
scrumtoys <web>
ejb31-war <ejb, web>
Command list-applications executed successfully.
```

This example specifies the `--host asadmin` utility option and the `--type` subcommand option with the `list-applications` subcommand in single mode. In this example, the DAS is running on the host `srvr1.example.com` and applications of type `web` are to be listed.

To Display Help Information for the `asadmin` Utility or a Subcommand

Payara Server provides help information about the syntax, purpose, and options of the `asadmin` utility and its subcommands.

This help information is also available in the "Payara Server Command Reference Manual".

1. If you are displaying help information for a remote subcommand, ensure that the server is running. Remote subcommands require a running server.
2. Specify the subcommand of interest as the operand of the `help` subcommand. If you run the `help` subcommand without an operand, help information for the `asadmin` utility is displayed.

Example 2-4 Displaying Help Information for the `asadmin` Utility

```
asadmin help
```

This example displays the help information for the `asadmin` utility.

Example 2-5 Displaying Help Information for an `asadmin` Utility Subcommand

```
asadmin help create-jdbc-resource
```

This example displays the help information for the `create-jdbc-resource` subcommand.

To display the available subcommands, use the "`list-commands`" subcommand. Local subcommands are displayed before remote subcommands. If the server is not running, only local subcommands are displayed.

To Start a Multimode Session

The `asadmin` utility can be used in multiple command mode, or multimode. In multimode, you run the `asadmin` utility once to start a multimode session.During the session, the `asadmin` utility continues to accept subcommands until you end the session and return to the operating system's command shell.

Any `asadmin` utility options that you set for your multimode session are used for all subsequent subcommands in the session.

NOTE	Starting a multimode session does not require a running DAS.

1. Do one of the following:
   - Run the `asadmin` utility without a subcommand.
   - Use the "`multimode`" subcommand.
2. If necessary, also specify any `asadmin` utility options that will apply throughout the multimode session.
3. In a multimode session, the `asadmin>` prompt is displayed on the command line. You can now type `asadmin` subcommands at this prompt to administer Payara Server.

Example 2-6 Starting a Multimode Session with `asadmin` Utility Options

```
asadmin --user admin1 --passwordfile pwd.txt multimode
```

This example starts a multimode session in which the `asadmin` utility options `--user` and `--passwordfile` are set for the session.

Example 2-7 Starting a Multimode Session by Using the `multimode` Subcommand

```
asadmin multimode
```

This example uses the `multimode` subcommand to start a multimode session in which the default `asadmin` utility options are used.

The `asadmin>` prompt is displayed on the command line.

Example 2-8 Running a Subcommand in a Multimode Session

```
asadmin
Use "exit" to exit and "help" for online help.

asadmin> list-domains
Name: domain1 Status: Running
Command list-domains executed successfully.
asadmin>
```

This example starts a multimode session and runs the `list-domains` subcommand in the session.

Starting a Multimode Session From Within an Existing Multimode Session

You can start a multimode session from within an existing session by running the `multimode` subcommand from within the existing session. After you end the second multimode session, you return to your original multimode session.

To End a Multimode Session

At the `asadmin>` prompt, type one of the following commands or key combinations:

- `exit`
- `quit`
- UNIX and Linux systems: `Ctrl-D`
- Windows systems: `Ctrl-Z`

CAUTION	Do not type `Ctrl-C` to end a multimode session. If a domain or Payara Server instance is started from the multimode session, typing

> `Ctrl-C` kills the domain or instance process.

You are returned to the operating system's command shell and the `asadmin>` prompt is no longer displayed. If the `asadmin>` prompt is still displayed, you might have opened a multimode session within a multimode session. In this situation, repeat this procedure to end the remaining multimode session.

To Enable Multimode Session Event Designators

This feature allows the use of **Bash Event Designators** when using Payara Server's Asadmin CLI in multimode.

An Event Designator is a reference to a command line entry in the history list, which makes it easy for users to refer and re-use previous asadmin subcommands entries.

IMPORTANT	This feature is disabled by default in Payara Server and has to be enabled first before being used.
WARNING	If this feature is enabled, all occurrences of the `\` character will be escaped and thus, removed before running the corresponding command. You'll have to escape these characters and turn them into `\\` for any commands to be properly parsed.

Enabling Jline Event Expansion

To enable support for JLine event designators just set the `AS_ADMIN_DISABLE_EVENT_EXPANSION` environment variable to `false`. Follow these steps depending on your operating system:

Linux and *Nix Variants

1. Open the Terminal
2. Type the following command

```
export AS_ADMIN_DISABLE_EVENT_EXPANSION=false
```

3. Check that the variable has been correctly set:

```
echo $AS_ADMIN_DISABLE_EVENT_EXPANSION
```

Windows

1. Open Command Prompt as an Administrator
2. Type the following commands

```
setx AS_ADMIN_DISABLE_EVENT_EXPANSION "false" /m
set AS_ADMIN_DISABLE_EVENT_EXPANSION=false
```

3. Check that the variable has been correctly set:

```
echo %AS_ADMIN_DISABLE_EVENT_EXPANSION%
```

To Run a Set of `asadmin` Subcommands from a File

Running a set of `asadmin` subcommands from a file enables you to automate repetitive tasks.

1. Create a plain text file that contains the sequence of subcommands that you want to run.
2. Run the "`multimode`" subcommand, specifying the file that you created.

   If necessary, also specify any `asadmin` utility options that are required to enable subcommands in the file to run.

Example 2-9 Running a Set of `asadmin` Subcommands from a File.

```
create-domain --portbase 9000 customdomain
start-domain customdomain
list-commands
stop-domain customdomain
delete-domain customdomain
```

This example contains the following:

- A listing of a file that is named `commands_file.txt`, which contains a sequence of

`asadmin` subcommands.

- The command to run the subcommands in the file `commands_file.txt`

The `commands_file.txt` file contains the `asadmin` utility subcommands to perform the following sequence of operations:

1. Creating the domain `customdomain`
2. Starting the domain `customdomain`
3. Listing all available subcommands
4. Stopping the domain `customdomain`
5. Deleting the domain `customdomain`

This example runs the sequence of subcommands in the `commands_file.txt` file. Because the `--portbase` option is specified for the `create-domain` subcommand in the file, the `--port asadmin` utility option must also be set.

```
asadmin --port 9048 multimode --file commands_file.txt
```

For more information about the subcommands in the preceding example, see the following help pages:

- "`create-domain`"
- "`delete-domain`"
- "`list-commands`"
- "`multimode`"
- "`start-domain`"
- "`stop-domain`"

To Run `asadmin` Subcommands in `--detach` Mode

You can use the `--detach` option of the `asadmin` utility to detach `asadmin` subcommands and run them in the background in detach mode. The `asadmin`

`--detach` option is useful for long-running subcommands and enables you to run several independent subcommands from one console or script.

1. Ensure that the server is running. Remote commands require a running server.
2. Detach and run the subcommand by using the `asadmin --detach` option.

Example 2-10 Using the `--detach` Option in Single Mode

```
asadmin --detach create-cluster Cluster1
Job ID: 1
Command create-cluster started successfully.
```

This example uses the `asadmin --detach` option in single mode to run the `create-cluster` subcommand.

Example 2-11 Using the `--detach` Option in Multimode

```
asadmin> create-cluster Cluster1 --detach
Job ID: 1
Command create-cluster started successfully.
```

This example uses the `asadmin --detach` option in multimode to run the `create-cluster` subcommand.

Job IDs are assigned to subcommands that are started using the `asadmin --detach` option or that contain progress information. You can use the `list-jobs` subcommand to list jobs and their job IDs, the `attach` subcommand to reattach to a job and view its status, and the `configure-managed-jobs` subcommand to configure how long information about jobs is kept.

Example 2-12 Listing Jobs

```
asadmin> list-jobs
JOB ID     COMMAND           STATE       EXIT CODE TIME OF COMPLETION
1          create-instance    COMPLETED   SUCCESS   2023-02-15 16:16:16 PST
2          deploy            COMPLETED   FAILURE   2023-02-15 18:26:30 PST
Command list-jobs executed successfully
```

This example runs the `list-jobs` subcommand in multimode to list jobs and job information.

Example 2-13 Attaching to a Subcommand and Checking Its Status

```
asadmin> attach 1
Command create-cluster executed with status SUCCESS.
Command attach executed successfully.
```

This example runs the `attach` subcommand in multimode to attach to the `create-cluster` subcommand with a job ID of `1`. If a subcommand is still in progress, the output displays the current status, such as percentage complete.

Example 2-14 Configuring Managed Jobs

```
asadmin> configure-managed-jobs --job-retention-period=36h
Command configure-managed-jobs executed successfully.
```

This example runs the `configure-managed-jobs` subcommand in multimode to set the job retention period to 36 hours. Time periods can be specified in `Hh|Mm|Ss` format for hours, minutes, or seconds.

For the full syntax and options of the subcommands in the preceding examples, see the following help pages:

- "`attach`"
- "`configure-managed-jobs`"
- "`list-jobs`"

Administering System Properties

Shared server instances will often need to override attributes defined in their referenced configuration. Any configuration attribute can be overridden through a system property of the corresponding name.

To Create System Properties

Use the `create-system-properties` subcommand in remote mode to create or update one or more system properties of the domain or configuration. Any configuration attribute can be overwritten through a system property of the corresponding name.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create system properties by using the "`create-system-properties`" subcommand.

Example 2-15 Creating a System Property

```
asadmin> create-system-properties http-listener-port=1088
Command create-system-properties executed successfully.
```

This example creates a system property associated with `http-listener-port=1088` on `localhost`.

To List System Properties

Use the `list-system-properties` subcommand in remote mode to list the system properties that apply to a domain, cluster, or server instance or configuration.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List system properties by using the "`list-system-properties`" subcommand.

The existing system properties are displayed, including predefined properties such as `HTTP_LISTENER_PORT` and `HTTP_SSL_LISTENER_PORT`.

Example 2-16 Listing System Properties

```
asadmin> list-system-properties
http-listener-port=1088
Command list-system-properties executed successfully.
```

This example lists the system properties on host `localhost`.

To Delete a System Property

Use the `delete-system-property` subcommand in remote mode to delete system properties.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the existing system properties by using the "`list-system-properties`" subcommand.

3. Delete the system property by using the "`delete-system-property`" subcommand.
4. If necessary, notify users that the system property has been deleted.

Example 2-17 Deleting a System Property

```
asadmin> delete-system-property http-listener-port
Command delete-system-property executed successfully.
```

This example deletes a system property named `http-listener-port` from `localhost`.

System Properties Inventory

The following properties are used exclusively by Payara Server to configure specific settings across multiple features:

`fish.payara.clientHttpsProtocol`: Sets the TLS version to be used by the asadmin client.

+ This is separate from the TLS version set for HTTPS communication on a listener.

+ Valid values are:

- `TLSv1.2`,
- `TLSv1.3`
- `TLSv1.3` (default)

`fish.payara.classloading.delegate`: When set to `false`, libraries from applications, and `domain-dir/lib` will override the library in the `as-install/glassfish/modules` directory

+ Defaults to `true`.

`fish.payara.admin.command.jline.log.disable`: When set to `true`, JLine logging entries for invalid key bindings and other unwanted errors will be prevented

+ NOTE: This is a property used exclusively by the Asadmin CLI

+ Defaults to `false`

`fish.payara.deployment.transform.namespace`: When set to `true`, namespace transformation (`javax.*` to `jakarta.\*` package refactoring) is always performed.

+ If the value is `undefined` then auto transformation is performed on the absence of `jakarta.\*` classes in the deployment archive otherwise transformation task is disabled.

+ Defaults to `null` (undefined)

`fish.payara.permittedFormBasedAuthHttpMethods`: A comma separated list of HTTP methods that are permitted to use for FORM based authentication (`j_security_check`).

+ If not set all HTTP methods are permitted.

+ Valid values are:

- `GET`
- `HEAD`
- `POST`
- `PUT`
- `DELETE`
- `CONNECT`
- `OPTIONS`
- `TRACE`
- `PATCH`

  Defaults to `null` (undefined)

`fish.payara.shutdowngrace`: After server shutdown is initiated the server waits this period of time (in seconds) to finish ongoing requests before terminating.

+ New requests are still accepted during this grace period. Accepts values between `0` - `Integer.MAX` and defaults to `0`.

`fish.payara.SO_KEEPALIVE`: Enables keep alive (`SO_KEEPALIVE`) on the sockets created by an IIOP listener. Can be set as a global system property, or as a property on a specific listener (with the latter taking precedence over the global setting if both are set). Defaults to `false`

`org.glassfish.grizzly.DEFAULT_MEMORY_MANAGER`: If option (defined as JVM Option with `-D` or as System Property value) `org.glassfish.grizzly.DEFAULT_MEMORY_MANAGER` is not present in the server/instance configuration, the server will override the Grizzly default and use the `HeapMemoryManager` class.

+ To override this behaviour (so to use a different memory manager), you must specify this JVM option in the server/instance config with your desired setting.

+ Valid values are:

- `org.glassfish.grizzly.memory.HeapMemoryManager` (default)
- `org.glassfish.grizzly.memory.PooledMemoryManager`
- `org.glassfish.grizzly.memory.ByteBufferManager`

`deployment.resource.validation`: This option (defined as JVM Option with `-D` or as System Property value) defines if the resources are validated during deployment of the application.

+ When active, the default value, all JNDI references are verified but those references can be provided by the application itself and thus not yet available at deployment time (for example the `java:module/env/` based ones).

+ By setting this option to `false`, validation is not performed and deployment will

succeed.

+ However, issues will occur at runtime of the application in case of errors in the references.

+ Defaults to `true`

`fish.payara.faces.integration.allowFacesCdiInitialisation`: This option specifies whether scanning for Jakarta Faces content during application deployment is enabled. When enabled, if Faces content is detected, it forces an attempt to initialise CDI and Weld, even if bean discovery is set to "none".

+ A similar property can be set as a property `deploy` command. See deploy for details.

+ The `deploy` command property has precedence over the value defined by this property.

+ Default is `false`.

`fish.payara.javax.servlet.jsp.disable-static-field-references`: This option brings back the behaviour of evaluating EL in JSP pages from previous iterations.

+ When enabled, a static field reference won't be evaluated.

+ Defaults to `false`

`fish.payara.node.ssh.timeout`: This option sets the SSH timeout in milliseconds for running commands on SSH nodes.

+ This is a separate timeout to the Asadmin CLI communication or any command specific timeout, this is specifically for the SSH connection used for running certain commands that operate on the remote node itself (e.g. `setup-ssh` and `start-instance`).

+ If set to an invalid value a warning will be logged and the default will be used.

+ Accepts values from `1` - `Integer.MAX`. Defaults to `120000ms`

`fish.payara.tenantcontrol.blocking.disable`: This option determines if a server instance should skip its standard checks for if it is available for a Hazelcast `Tenantable` object.

+ Defaults to `true`.

ifeval: ["enterprise" == "community"] `fish.payara.autoscale.maxscale`: This option sets the default maximum boundary for the number of instances to scale up or down by when using auto-scale groups.

+ Accepts values from `1` - `Integer.MAX`. Defaults to `100` instances. **endif**: []

ifeval: ["enterprise" == "enterprise"] `fish.payara.upgrade.repo.url`: This option defines the Maven repository URL used by the `upgrade-server` command.

+ Accepts a URL pointing to the Payara Enterprise Nexus repository to download from (expected to point to the `fish/payara/distributions` directory of the repository).

+ Defaults to `https://nexus.payara.fish/repository/payara-enterprise-downloadable-artifacts/fish/payara/distributions/` **endif**: []

`hazelcast.cp-subsystem.auto-promote`: When the Payara Server's Data Grid/Hazelcast CP subsystem is enabled (more than `0` members are configured), this option will enable or disable an auto-promote behaviour that will allow other server instances to join the Hazelcast CP subsystem when detected by the DAS.

+ IMPORTANT: The CP subsystem is automatically enabled on the Data Grid/Hazelcast when the `hazelcast.cp-subsystem.cp-member-count` system property is set an integer greater than zero.

+ Defaults to `true`

`hazelcast.auto-partition-group`: Enables/disables Hazelcast group auto-partitioning behaviour in Payara Server's Data Grid.

+ When auto-partitioning is turned on, instances that join the Data Grid will not follow

the default partition group configuration strategy and instead will form individual partition groups. Read more about how partition groups work in the official Hazelcast Platform documentation.

+ This option is intended to make member partitioning work consistently for highly dynamic environments (like Kubernetes).

+ CAUTION: Use this option with extreme care

ifeval: ["enterprise" == "community"] **`fish.payara.ear-combined-cdi-deployment`**: This option defines whether or not to restrict CDI scanning when multiple WARs exist within an EAR.

+ This option is intended to allow a user to revert the CDI scanning behaviour to how it operated in prior versions.

+ The behaviour in prior versions aligned with `false`.

+ Default is undefined (`true`) **endif**: []

`fish.payara.ready-after-applications`: This option determines when the server is marked as "ready".

+ When enabled: The server is marked as ready after all post-boot configuration has been applied and all previously deployed applications have been loaded.

+ When disabled: the server is marked as ready after all `InstanceLifecycleListener` components and services have processed the `server_startup` event, but before post-boot configuration is applied and previously deployed applications are loaded.

+ Defaults to `true`.

Using Configuration Modularity

With configuration modularity in Payara Server, new modules can be added to Payara Server distributions without modifying the global `domain.xml` configuration file.

Default configuration data for modules is stored in the modules themselves, rather than in `domain.xml`, and loaded when needed.

Module configuration elements are stored in `domain.xml` only when the default configuration included in the module is changed or when module configuration elements are added to `domain.xml` using the `create-module-config` subcommand. The `delete-module-config` subcommand removes module configuration elements from `domain.xml`, and the `get-active-module-config` subcommand displays the current active configuration of a module.

To Add the Default Configuration of a Module to `domain.xml`

Use the `create-module-config` subcommand to add the default configuration of a module to `domain.xml`.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Add the default configuration of a module to `domain.xml` by using "`create-module-config`" subcommand.

Example 2-18 Adding Module Configuration to `domain.xml`

```
asadmin> create-module-config web-container
Command create-module-config executed successfully.
```

This example adds the default configuration of the web container module to `domain1` in `server-config` (the default configuration). Use the `--dryrun` option to preview the configuration before it is added.

To Remove the Configuration of a Module from `domain.xml`

Use the `delete-module-config` subcommand to remove the configuration of a module from `domain.xml` and cause the module to use the default configuration included in the module.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Remove the configuration of a module from `domain.xml` by using the "`delete-module-config`" subcommand.

Example 2-19 Removing Module Configuration from `domain.xml`

```
asadmin> delete-module-config web-container
Command delete-module-config executed successfully.
```

This example deletes the configuration of the web container module from `domain1` in `server-config` (the default configuration).

To Display the Current Active Configuration of a Module

Use the `get-active-module-config` subcommand to display the current active configuration of a module.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Display the current active configuration of a module by using the "`get-active-module-config`" subcommand.

Example 2-20 Displaying the Current Active Configuration of a Module

```
asadmin> get-active-module-config jms-service
At location: domain/configs/config[server-config]
<jms-service default-jms-host="default_JMS_host" type="EMBEDDED"
  <jms-host port="7676" host="localhost" name="default_JMS_host"/>
</jms-service>
Command get-active-module-config executed successfully.
```

This example displays the current active configuration of the JMS service in `server-config` (the default configuration).

Administering Resources

This section contains instructions for integrating resources into the Payara Server environment.Information about administering specific resources, such as JDBC, is contained in other chapters.

To Add Resources from an XML File

Use the `add-resources` subcommand in remote mode to create the resources named in the specified XML file. The following resources are supported: JDBC connection pool and resource, JMS, JNDI, and JavaMail resources, custom resource, connector resource and work security map, admin object, and resource adapter configuration.

The XML file must reside in the `domain-dir/config` directory. If you specify a relative path or simply provide the name of the XML file, this subcommand will prepend `domain-dir/config` to this operand.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Add resources from an XML file by using the "`add-resources`" subcommand. Information about properties for the subcommand is included in this help page.
3. Restart Payara Server. See to-restart-a-domain.

Example 2-21 Adding Resources

```
asadmin> add-resources c:\tmp\resource.xml
Command : JDBC resource jdbc1 created successfully.
Command : JDBC connection pool poolA created successfully.
Command add-resources executed successfully.
```

This example creates resources using the contents of the `resource.xml` file on `localhost`.

Listing Various System Elements

To Display the Payara Server Version

Use the `version` subcommand in remote mode to display information about the Payara Server version for a particular server. If the subcommand cannot communicate with the server by using the specified login (user/password) and target (host/port) information, then the local version is displayed along with a warning message.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Display the version by using the "`version`" subcommand.

Example 2-22 Displaying Version Information

```
asadmin> version
Version = Payara Server 7.2.0 #badassfish (build 00)
Command version executed successfully.
```

This example displays the version of Payara Server on the local host.

To List Applications

Use the `list-applications` subcommand in remote mode to list the deployed applications. If the `--type` option is not specified, all applications are listed.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List applications by using the "`list-applications`" subcommand.

Example 2-23 Listing Applications

```
asadmin> list-applications --type web
hellojsp <web>
Command list-applications executed successfully.
```

This example lists the web applications on `localhost`.

To List Containers

Use the `list-containers` subcommand in remote mode to list application containers.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List containers by using the "`list-containers`" subcommand.

Example 2-24 Listing Containers

```
asadmin> list-containers
List all known application containers

faulttolerance
metrics
appclient
jpa
jpa
osgi
connector
grizzly
health
jwtauth
ejb
openapi
webservices
security
payara-resources_ear
payara-resources
resources_ear
resources
web
```

```
ear
cdi
cdi
config
Command list-containers executed successfully.
```

This example lists the containers on `localhost`.

To List Modules

Use the `list-modules` subcommand in remote mode to list the modules that are accessible to the Payara Server module subsystem. The status of each module is included. Possible statuses include NEW and READY.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List modules by using the "`list-modules`" subcommand.

Example 2-25 Listing Modules

```
asadmin> list-modules
```

This example lists the accessible modules.

Information similar to the following is displayed (partial output):

```
List Of Modules
Module Status Report Begins
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.naming [380]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.grizzly.nucleus-grizzly-all
[292]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.phonehome-
bootstrap [332]], State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.shell [401]], State = [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.osgi-resource-locator [1]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.connectors.internal-api
[51]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.asadmin-audit
[19]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.docker [78]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.security.ee [351]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-
modules.requesttracing-core [337]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.batch.glassfish-batch-
connector [109]], State = [READY]
```

```
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.security [353]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.deployment.dol [79]], State
= [READY]
OSGiModuleImpl:: Bundle = [org.objectweb.asm.tree.analysis [21]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.healthcheck-
core [132]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-metrics [276]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.jmx-monitoring [240]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.orb.enabler [299]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.core.kernel [251]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.healthcheck-checker [131]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.util [6]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.resources.connector [338]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.cli [4]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.payara-
executor-service [319]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.deployment.common [74]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.connectors.runtime [52]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.notification-jms-core [291]], State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.configadmin [394]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.transaction.internal-api
[362]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.concurrent.connector [45]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.internal-api [147]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jms.core [236]], State =
[READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.api [139]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.weld-integration [389]],
State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.gogo.shell [399]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.grizzly.glassfish-grizzly-
extra-all [116]], State = [READY]
OSGiModuleImpl:: Bundle = [org.objectweb.asm.commons [22]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.hk2.config-types [48]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.concurrent.impl [46]], State
= [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.config-api [47]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.notification-
cdi-eventbus-core [288]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-openapi [278]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.gf-admingui-
connector [96]], State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.eventadmin [395]], State = [READY]
```

```
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.hazelcast-
bootstrap [127]], State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.gogo.command [397]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.util [44]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-config-extensions [264]], State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.scr [400]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.gf-web-connector [105]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.glassfish-mbeanserver
[118]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.gf-weld-connector
[106]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.environment-warning [88]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.opentracing-
adapter [295]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.healthcheck-metrics [134]], State = [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.runlevel [144]], State = [READY]
OSGiModuleImpl:: Bundle =
[fish.payara.server.internal.resourcebase.resources.nucleus-resources [293]], State
= [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.utils [145]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.rest-service [344]],
State = [READY]
OSGiModuleImpl:: Bundle = [org.objectweb.asm.util [24]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-fault-tolerance [270]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.monitoring-core
[286]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.transaction.jts [250]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.load-balancer.gf-load-
balancer-connector [103]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.healthcheck-
cpool [133]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.amx-core [7]], State
= [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.core.glassfish [121]], State
= [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jdbc.config [207]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.security.services [352]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.microprofile-
config-service [265]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ejb.http-admin [83]], State
= [READY]
OSGiModuleImpl:: Bundle = [jakarta.inject.jakarta.inject-api [172]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ha.hazelcast-store [126]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.notification-
eventbus-core [290]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.payara-micro-service [324]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.transaction.jta [249]],
State = [READY]
```

```
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.notification-
core [289]], State = [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.locator [143]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.core.jakartaee-kernel
[195]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.rest-monitoring-service [343]], State = [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.class-model [31]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.container-common
[68]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ejb.gf-ejb-connector [99]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.core.logging [258]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.hk2.config [140]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.deployment.autodeploy [72]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.healthcheck-
stuck [136]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.glassfish-naming
[119]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-healthcheck [272]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.asadmin-
recorder [20]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.flashlight.framework [95]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.hazelcast-eclipselink-coordination [128]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.resources.javamail-connector
[198]], State = [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.osgi-adapter [313]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.webservices.connector
[384]], State = [READY]
OSGiModuleImpl:: Bundle = [org.objectweb.asm [25]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.connectors.gf-connectors-
connector [98]], State = [READY]
OSGiModuleImpl:: Bundle = [org.objectweb.asm.tree [23]], State = [READY]
OSGiModuleImpl:: Bundle =
[fish.payara.server.internal.security.jacc.provider.inmemory [150]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jms.gf-jms-connector [100]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jdbc.runtime [208]], State =
[READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.fileinstall [396]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.glassfish-api [107]],
State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.gogo.runtime [398]], State = [READY]
OSGiModuleImpl:: Bundle = [org.eclipse.angus.mail [10]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.gf-restadmin-connector
[104]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.webservices.jsr109-impl
[247]], State = [READY]

...
Command list-modules executed successfully.
```

To List Subcommands

Use the `list-commands` subcommand in remote mode to list the deployed `asadmin` subcommands. You can specify that only remote subcommands or only local subcommands are listed. By default, this subcommand displays a list of local subcommands followed by a list of remote subcommands.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List subcommands by using the "`list-commands`" subcommand.

Example 2-26 Listing Subcommands

```
asadmin> list-commands --localonly
create-domain
delete-domain
list-commands
list-domains
login
monitor
start-database
start-domain
stop-domain
stop-database
version
Command list-commands executed successfully.
```

This example lists only local subcommands.

To List Timers

The timer service is a persistent and transactional notification service that is provided by the enterprise bean container and is used to schedule notifications or events used by enterprise beans. All enterprise beans except stateful session beans can receive notifications from the timer service.

Persistent timers set by the service are not destroyed when the server is shut down or restarted.

Use the `list-timers` subcommand in remote mode to list the persistent timers owned by a specific server instance. You can use this information to decide whether to do a timer migration, or to verify that a migration has been completed successfully.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List timers by using the "`list-timers`" subcommand.

Example 2-27 Listing Timers

```
asadmin> list-timers server
1
The list-timers command was executed successfully.
```

This example lists the timers in a particular standalone server instance. There is one currently active timer set.

To Show Component Status

Use the `show-component-status` subcommand in remote mode to get the status (either enabled or disabled) of the specified deployed component.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Show component status by using the "`show-component-status`" subcommand.

Example 2-28 Showing Status of a Component

```
asadmin> show-component-status MEjbApp
Status of MEjbApp is enabled
Command show-component-status executed successfully.
```

This example shows the status of the `MEjbApp` component.

- GNU Wget

The implementation of the Payara Server REST interfaces is based on project Jersey. Project Jersey is the reference implementation of Jakarta RESTful Web Services.

See Also

- Event Designators

Using REST Interfaces to Administer Payara Server

Payara Server provides representational state transfer (REST) interfaces to enable you to access monitoring and configuration data for Payara Server, including data that is

provided by newly installed add-on components.

You can access the Payara Server REST interfaces through client applications such as:

- Web browsers
- REST API Clients
- cURL

It allows execution of administration commands in a similar way as the `asadmin` utility, such as:

- Execute administration commands to modify the domain's configuration.
- Retrieve monitoring statistics
- Look at the domain's log entries.

Using REST Urls to Administer Payara Server

Each object in the configuration and monitoring object trees is represented as a REST resource that is accessible through an HTTP uniform resource locator (URL).

Access to REST resources for Payara Server monitoring and configuration data requires a running DAS.

REST Urls to Resources for Configuration and Monitoring Objects

The formats of the URLs to resources that represent objects in the configuration and monitoring object trees are as follows:

- Configuration: `http://host:port/management/domain/path`
- Monitoring: `http://host:port/monitoring/domain/path`

The replaceable items in these URLs are as follows:

host: The host where the DAS is running. **port**: The HTTP port or HTTPS port for administration. **path**: The path to the object. The path is the dotted name of the object in which each dot (`.`) is replaced with a slash (`/`).

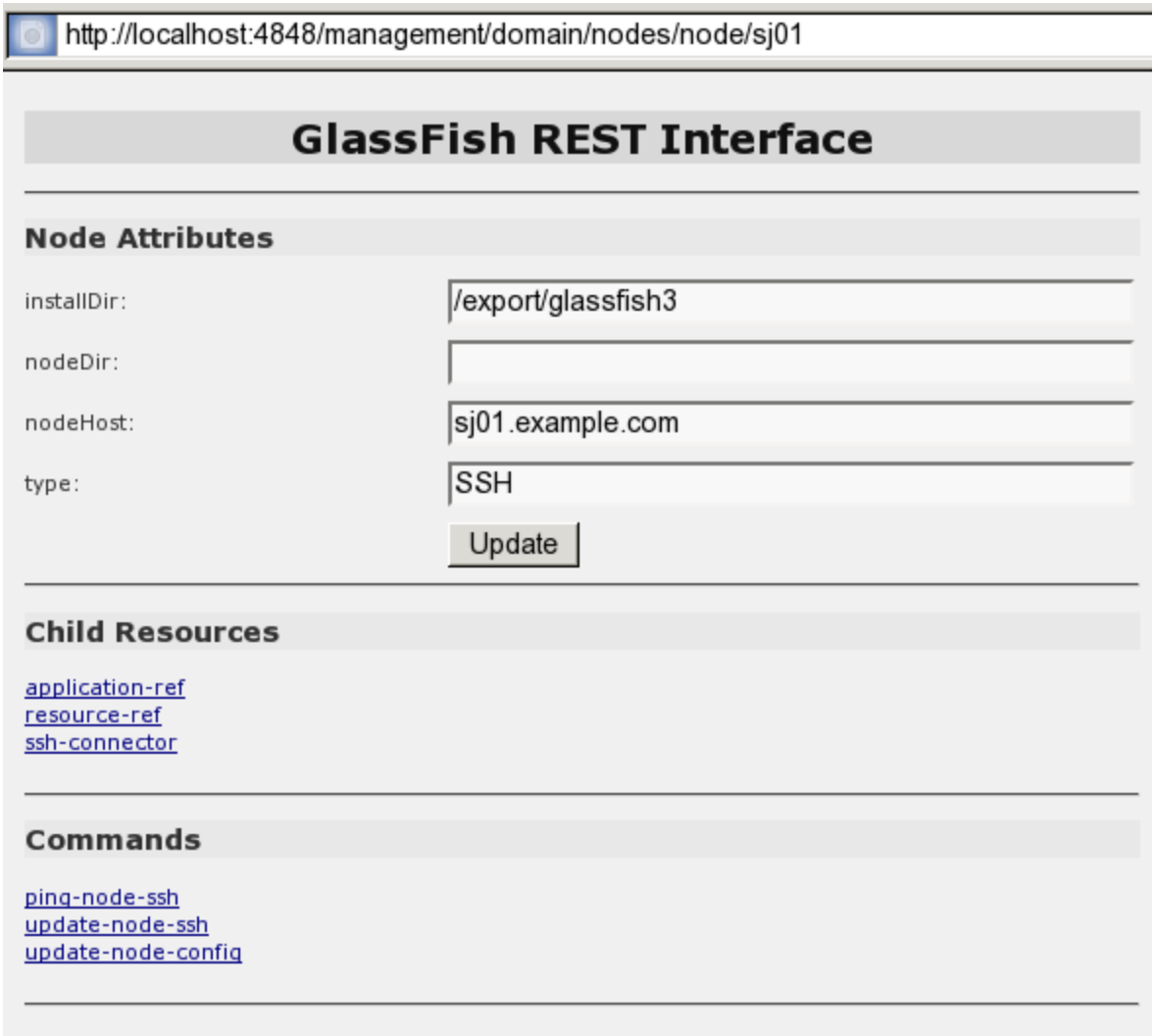
+ NOTE: The path to a Payara Server instance is `servers/server/<instance-name>`, where `instance-name` is the name of the instance. For the DAS, `instance-name` is `server` and the path is `servers/server/server`.

If the URL to a REST resource for Payara Server monitoring or configuration data is opened in a web browser, the browser displays a web page that contains the following information about the resource:

- A list of the attributes of the resource and their values. If the resource represents an object in the configuration tree, these attributes are presented in an HTML form that you can use to update the resource. Attributes of a resource for an object in the monitoring tree are read only.
- A list of hypertext links to the children of the resource. This list of links enables you to traverse the tree that contains the resource and to discover the all resources in the tree.
- A list of hypertext links to resources that represent `asadmin` subcommands for non-CRUD operations on the resource.

The following figure shows the web page for the REST resource for managing a domain.

Web Page for the REST Resource for Managing a Domain



REST Urls for Accessing the Log File

The `server.log` file of the DAS is represented as a child that is named `view-log` of the resource for managing the domain. A child of the resource for the `server.log` file represents the log file details

The formats of the URLs to resources that represent the log file are as follows:

- Log file: `http://host:port/management/domain/view-log`
- Log file details: `http://host:port/monitoring/domain/view-log/details`

The replaceable items in these URLs are as follows:

host: The host where the DAS is running. **port**: The HTTP port or HTTPS port for administration.

You can use the optional `start` parameter in the URL to the resource for the log file to specify the number of characters at the start of the file to skip.

For example, to skip 10,000 characters, specify the URL as `http://localhost:4848/management/domain/view-log?start=10000`. This example assumes that the DAS is running on the local host and uses the default port for administration.

The resource for the log file returns the HTTP header `X-Text-Append-Next`, which contains the entire URL to pass to the `GET` method to return the changes since the last call. You can use this header in client applications to get all log entries that were added in particular interval.

For example, by testing the value of the `X-Text-Append-Next` header in a client thread every 10 seconds, you can monitor the log entries that were added in the last 10 seconds.

Using REST Resource Methods to Administer Payara Server

The Payara Server REST interfaces support methods for accessing objects in the monitoring and configuration object trees.

The following table shows the REST methods for administering monitoring and configuration data and the tasks that you can perform with each method.

Table 2. REST Resource Methods for Administering Monitoring and Configuration Data

Task	REST Method
Determine the methods and method parameters that an object in the tree supports	`GET`
Retrieve data for an object in the tree	`GET`
Add an object to the tree	`POST`
Update an object in the tree	`POST`
Delete an object from the tree	`DELETE`

NOTE REST requests that add, update, or delete objects must specify the `X-Requested-By` header with the value `GlassFish REST HTML interface`.

The `GET` method determines the methods and method parameters that an object in the tree supports and provides additional information about the object. For details, see to-retrieve-data-for-an-object-in-the-tree.

To Determine the Methods and Method Parameters that an Object in the Tree Supports

The methods and method parameters that an object in the tree supports depend on the REST resource that represents the object:

- REST resources for monitoring support only the `GET` method.
- All REST resources for configuration support the `GET` method.

WARNING However, only some REST resources for configuration also support the `POST` method and the `DELETE` method.

Before performing any operations on an object in the tree, determine the methods and method parameters that the object supports.

You can specify the format in which this information is presented. For more information, see formats-for-resource-representation-of-configuration-objects.

NOTE Each `POST` method and `DELETE` method that a REST resource supports has an equivalent `asadmin` subcommand. The parameters of a `POST` method or a `DELETE` method correspond to the options of the method's equivalent `asadmin` subcommand.

1. Ensure that the server is running. Operations on REST resources for Payara Server data require a running server.
2. Use the `GET` method on the REST resource that represents the object. The `GET` method returns the list of methods that the resource supports. For each method, the

list of acceptable message parameters or the list of acceptable query parameters are returned.

This example uses the cURL utility to determine the methods and method parameters that the resource for the node `sj01` supports.

The example uses the following options of the cURL utility:

- `-X` to specify that the `GET` method is used
- `-H` to specify that the resource is represented in JavaScript Object Notation (JSON)

In this example, the DAS is running on the local host and the HTTP port for administration is 4848. The resource supports the `GET` method and the `POST` method.

```
curl -X GET -H "Accept: application/json"
http://localhost:4848/management/domain/nodes/node/sj01

{
  "command":"Node",
  "exit_code":"SUCCESS",
  "extraProperties":{
    "commands":[
      {"path":"_delete-node","command":"delete-node","method":"DELETE"},
      {"path":"_update-node","command":"_update-node","method":"POST"},
      {"path":"ping-node-ssh","command":"ping-node-ssh","method":"GET"},
      {"path":"update-node-ssh","command":"update-node-ssh","method":"POST"},
      {"path":"update-node-config","command":"update-node-config","method":"POST"
}],
    "methods":[
      {"name":"GET"},
      {"name":"POST","messageParameters":{
        "installDir":{"optional":"true","type":"string","key":"false"},
        "nodeDir":{"optional":"true","type":"string","key":"false"},
        "nodeHost":{"optional":"true","type":"string","key":"false"},
        "type":{"optional":"true","type":"string","key":"false"}
        }
      }
    ],
    "entity":{
      "installDir":"\/export\/payara7",
      "name":"sj01",
      "nodeDir":null,
      "nodeHost":
      "sj01.example.com",
      "type":"SSH"
    },
    "childResources":{
      "application-ref":
        "https:\/\/localhost:4848\/management\/domain\/nodes\/node\/sj01
\/application-ref",
```

```
      "resource-ref":
        "https:\/\/localhost:4848\/management\/domain\/nodes\/node\/sj01\/resource-
ref",
      "ssh-connector":
        "https:\/\/localhost:4848\/management\/domain\/nodes\/node\/sj01\/ssh-
connector"
    }
  }
}
```

To Retrieve Data for an Object in the Tree

Retrieving data for an object in the tree obtains the following information about the REST resource that represents the object:

- A list of the REST methods that the resource supports
- A list of the attributes of the resource and their values
- A list of URLs to the children of the resource

You can specify the format in which this information is presented. For more information, see formats-for-resource-representation-of-configuration-objects.

1. Ensure that the server is running. Operations on REST resources for Payara Server data require a running server.
2. Use the `GET` method on the REST resource that represents the object.

This example uses the cURL utility to retrieve data for the resource for a the node `sj01`. The example uses the following options of the cURL utility:

- `-X` to specify that the `GET` method is used
- `-H` to specify that the resource is represented in JavaScript Object Notation (JSON)

In this example, the DAS is running on the local host and the HTTP port for administration is `4848`.

```
curl -X GET -H "Accept: application/json"
http://localhost:4848/management/domain/nodes/node/sj01
{
  "command":"Node",
  "exit_code":"SUCCESS",
  "extraProperties":{
```

```
    "commands":[
      {"path":"_delete-node","command":"delete-node","method":"DELETE"},
      {"path":"_update-node","command":"_update-node","method":"POST"},
      {"path":"ping-node-ssh","command":"ping-node-ssh","method":"GET"},
      {"path":"update-node-ssh","command":"update-node-ssh","method":"POST"},
      {"path":"update-node-config","command":"update-node-config","method":"POST"
}],
    "methods":[
      {"name":"GET"},
      {"name":"POST","messageParameters":{
        "installDir":{"optional":"true","type":"string","key":"false"},
        "nodeDir":{"optional":"true","type":"string","key":"false"},
        "nodeHost":{"optional":"true","type":"string","key":"false"},
        "type":{"optional":"true","type":"string","key":"false"}
        }
      }
    ],
    "entity":{
      "installDir":"\/export\/payara7",
      "name":"sj01",
      "nodeDir":null,
      "nodeHost":
      "sj01.example.com",
      "type":"SSH"
    },
    "childResources":{
      "application-ref":
        "https:\/\/localhost:4848\/management\/domain\/nodes\/node\/sj01
\/application-ref",
      "resource-ref":
        "https:\/\/localhost:4848\/management\/domain\/nodes\/node\/sj01\/resource-
ref",
      "ssh-connector":
        "https:\/\/localhost:4848\/management\/domain\/nodes\/node\/sj01\/ssh-
connector"
    }
  }
}
```


To Add an Object to the Tree

1. Ensure that the server is running. Operations on REST resources for Payara Server data require a running server.
2. Determine the acceptable message parameters for the `POST` method of the resource that represents the parent of the object. For information about how to perform this step, see to-determine-the-methods-and-method-parameters-that-an-object-in-the-tree-supports.
3. Use the `POST` method on the REST resource that represents the parent of the object that you are adding.

4. Confirm that the object has been added. Perform this step on the resource that represents the object that you have just added, not the parent.

   For information about how to perform this step, see to-retrieve-data-for-an-object-in-the-tree.

This example uses the cURL utility to add a JDBC resource object to the tree by creating a REST resource to represent the JDBC resource.

In this example, the DAS is running on the local host and the HTTP port for administration is `4848`.

1. This step determines the acceptable message parameters for the `POST` method of the resource `jdbc-resource`.

```
curl -X GET -H "Accept: application/json"

http://localhost:4848/management/domain/resources/jdbc-resource
{
  "command":"Jdbc-resource",
  "exit_code":"SUCCESS",
  "extraProperties":{
    "commands":[],
    "methods":[
      {"name":"GET"},
      {"name":"POST","messageParameters":{
        "description":{"acceptableValues":"","optional":"true","type":"string"
,"defaultValue":""},
        "enabled":{"acceptableValues":"",optional":"true","type":"
boolean",defaultValue":"true"},
        "id":{"acceptableValues":"","optional":"false","type":"string"
,"defaultValue":""},
        "poolName":{"acceptableValues":"","optional":"false","type":"string"
,"defaultValue":""},
        "property":{"acceptableValues":"","optional":"true","type":"string"
,"defaultValue":"},
        "target":{"acceptableValues":"","optional":"true","type":"string
","defaultValue":""}
        }
      }
    ],
    "childResources":{
      "jdbc\/__TimerPool":
        "https:\/\/localhost:4848\/management\/domain\/resources\/jdbc-
resource\/jdbc%2F__TimerPool",
      "jdbc\/__default":
        "https:\/\/localhost:4848\/management\/domain\/resources\/jdbc-
resource\/jdbc%2F__default"
    }
  }
```

```
}
```

2. This step adds a resource as a child of the `jdbc-resource` resource. The `-d` option of the cURL utility sets the required message parameters as follows:
   - `id` is set to `jdbc/myjdbcresource`.
   - `connectionpoolid` is set to `DerbyPool`.

   ```
   curl -X POST -H "X-Requested-By: GlassFish REST HTML interface" -d id
   =jdbc/myjdbcresource -d connectionpoolid=DerbyPool
   http://localhost:4848/management/domain/resources/jdbc-resource
   ```

3. This step confirms that the object has been added by retrieving data for the REST resource that represents the object.

```
curl -X GET -H "Accept: application/json"
http://localhost:4848/management/domain/resources/
jdbc-resource/jdbc%2Fmyjdbcresource
{
  "command":"Jdbc-resource",
  "exit_code":"SUCCESS",
  "extraProperties":{
    "commands":[],
    "methods":[
      {"name":"GET"},
      {"name":"POST","messageParameters":{
        "description":{"optional":"true","type":"string","key":"false"},
        "enabled":{"optional":"true","type":"boolean","defaultValue":"true"
,"key":"false"},
        "jndiName":{"optional":"true","type":"string","key":"true"},
        "objectType":{"optional":"true","type":"string","defaultValue":"user"
,"key":"false"},
        "poolName":{"optional":"true","type":"string","key":"false"}
        }
      },
      {"name":"DELETE","messageParameters":{
        "target":{"acceptableValues":"","optional":"true","type":"string"
,"defaultValue":""}
        }
      }
    ],
    "childResources":{
      "property":
        "https:\/\/localhost:4848\/management\/domain\/resources\/jdbc-
resource\/jdbc%2Fmyjdbcresource\/property"
    }
  }
}
```

To Update an Object in the Tree

1. Ensure that the server is running.

   Operations on REST resources for Payara Server data require a running server.

2. Determine the acceptable message parameters for the `POST` method of the resource that represents the object. For information about how to perform this step, see to-determine-the-methods-and-method-parameters-that-an-object-in-the-tree-supports.
3. Use the `POST` method on the REST resource that represents the object that you are updating.
4. Confirm that the object has been updated. For information about how to perform this step, see to-retrieve-data-for-an-object-in-the-tree.

This example uses the cURL utility to update a JDBC resource in the tree by modifying the REST resource that represents the JDBC resource.

In this example, the DAS is running on the local host and the HTTP port for administration is `4848`.

1. This step determines the acceptable message parameters for the `POST` method of the resource `jdbc-myjdbcresource`.

```
curl -X OPTIONS -H "Accept: application/json"
http://localhost:4848/management/domain/resources/jdbc-resource/jdbc-
myjdbcresource

{
  "command":"Jdbc-resource",
  "exit_code":"SUCCESS",
  "extraProperties":{
    "commands":[],
    "methods":[
      {"name":"GET"},
      {"name":"POST","messageParameters":{
        "description":{"optional":"true","type":"string","key":"false"},
        "enabled":{"optional":"true","type":"boolean","defaultValue":"true"
,"key":"false"},
        "jndiName":{"optional":"true","type":"string","key":"true"},
        "objectType":{"optional":"true","type":"string","defaultValue":"user"
,"key":"false"},
        "poolName":{"optional":"true","type":"string","key":"false"}
        }
      },
```

```
      {"name":"DELETE","messageParameters":{
        "target":{"acceptableValues":"","optional":"true","type":"string"
,"defaultValue":""}
        }
      }
    ],
    "childResources":{
      "property":
        "https:\/\/localhost:4848\/management\/domain\/resources\/jdbc-
resource\/jdbc%2Fmyjdbcresource\/property"
    }
  }
}
```

2. This step updates the REST resource `jdbc-myjdbcresource` to disable the JDBC resource that `jdbc-myjdbcresource` represents.

The `-d` option of the cURL utility sets the `enabled` message parameter to `disabled`.

+

```
curl -X POST -H "X-Requested-By: GlassFish REST HTML interface" -d "enabled=false"
http://localhost:4848/management/domain/resources/jdbc-
resource/jdbc%2Fmyjdbcresource
```

1. This step confirms that the object has been updated by retrieving data for the REST resource that represents the object.

```
curl -X GET -H "Accept: application/json"
http://localhost:4848/management/domain/resources/
jdbc-resource/jdbc%2Fmyjdbcresource
{
  "command":"Jdbc-resource",
  "exit_code":"SUCCESS",
  "extraProperties":{
    "commands":[],
    "methods":[
      {"name":"GET"},
      {"name":"POST","messageParameters":{
        "description":{"optional":"true","type":"string","key":"false"},
        "enabled":{"optional":"true","type":"boolean","defaultValue":"true"
,"key":"false"},
        "jndiName":{"optional":"true","type":"string","key":"true"},
        "objectType":{"optional":"true","type":"string","defaultValue":
        "user","key":"false"},
        "poolName":{"optional":"true","type":"string","key":"false"}
        }
      },
      {"name":"DELETE","messageParameters":{
        "target":{"acceptableValues":"","optional":"true","type":"string"
,"defaultValue":""}
```

```
          }
        }
      ],
    "entity":{
      "description":null,
      "enabled":"false",
      "jndiName":"jdbc\/myjdbcresource",
      "objectType":
      "user",
      "poolName":"DerbyPool"
    },
    "childResources":{
      "property":
        "https:\/\/localhost:4848\/management\/domain\/resources\/jdbc-resource\/
         jdbc%2Fmyjdbcresource\/property"
      }
    }
}
```

To Delete an Object from the Tree

1. Ensure that the server is running. Operations on REST resources for Payara Server data require a running server.
2. Confirm that the object can be deleted. For information about how to perform this step, see to-determine-the-methods-and-method-parameters-that-an-object-in-the-tree-supports.
3. Confirm that the object has been deleted. Perform this step on the resource that represents the parent of the object that you have just deleted.

For information about how to perform this step, see to-retrieve-data-for-an-object-in-the-tree.

This example uses the cURL utility to delete a JDBC resource from the tree by deleting the REST resource that represents the JDBC resource.

In this example, the DAS is running on the local host and the HTTP port for administration is `4848`.

1. This step confirms that the object can be deleted by retrieving the REST methods that the resource `jdbc-myjdbcresource` supports.

Line breaks and white space are added to enhance readability.

```
curl -X GET -H "Accept: application/json"
http://localhost:4848/management/domain/resources/jdbc-
resource/jdbc%2Fmyjdbcresource

{
  "command":"Jdbc-resource",
  "exit_code":"SUCCESS",
  "extraProperties":{
    "commands":[],
    "methods":[
      {"name":"GET"},
      {"name":"POST","messageParameters":{
        "description":{"optional":"true","type":"string","key":"false"},
        "enabled":{"optional":"true","type":"boolean","defaultValue":"true"
,"key":"false"},
        "jndiName":{"optional":"true","type":"string","key":"true"},
        "objectType":{"optional":"true","type":"string","defaultValue":"user"
,"key":"false"},
        "poolName":{"optional":"true","type":"string","key":"false"}
        }
      },
      {"name":"DELETE","messageParameters":{
        "target":{"acceptableValues":"","optional":"true","type":"string"
,"defaultValue":""}
        }
      }
    ],
    "childResources":{
      "property":
        "https:\/\/localhost:4848\/management\/domain\/resources\/jdbc-
resource\/
        jdbc%2Fmyjdbcresource\/property"
    }
  }
}
```

2. This step deletes the `jdbc/myjdbcresource` resource.

```
curl -X DELETE -H "X-Requested-By: GlassFish REST HTML interface"
http://localhost:4848/management/domain/resources/jdbc-
resource/jdbc%2Fmyjdbcresource
```

3. This step confirms that the object has been deleted by retrieving data for the REST resource that represents the parent of the object.

```
curl -X GET -H "Accept: application/json"
http://localhost:4848/management/domain/resources/jdbc-resource

{
  "command":"Jdbc-resource",
  "exit_code":"SUCCESS",
```

```
  "extraProperties":{
    "commands":[],
    "methods":[
      {"name":"GET"},
      {"name":"POST","messageParameters":{
        "description":{"acceptableValues":"","optional":"true","type":"string"
,"defaultValue":""},
        "enabled":{"acceptableValues":"",optional":"true","type":"
boolean",defaultValue":"true"},
        "id":{"acceptableValues":"","optional":"false","type":"string"
,"defaultValue":""},
        "poolName":{"acceptableValues":"","optional":"false","type":"string"
,"defaultValue":""},
        "property":{"acceptableValues":"","optional":"true","type":"string"
,"defaultValue":"},
        "target":{"acceptableValues":"","optional":"true","type":"string
","defaultValue":""}
        }
      }
    ],
    "childResources":{
      "jdbc\/__TimerPool":
        "https:\/\/localhost:4848\/management\/domain\/resources\/jdbc-
resource\/jdbc%2F__TimerPool",
      "jdbc\/__default":
        "https:\/\/localhost:4848\/management\/domain\/resources\/jdbc-
resource\/jdbc%2F__default"
    }
  }
}
```


Resources for `asadmin` Subcommands that Perform Non-CRUD Operations

The Payara Server REST interfaces also support operations other than create, read, update, and delete (CRUD) operations, for example:

- State Management
- Queries
- Application deployment

These operations are supported through REST resources that represent the `asadmin` subcommands for performing these operations. Each resource is a child of the resource on which the operation is performed. The child resources do not represent objects in the configuration object tree.

For example, the resource that represents a node provides child resources for the following `asadmin` subcommands that perform non-CRUD operations on the node:

- `ping-node-ssh`
- `update-node-config`
- `update-node-ssh`

Securing Payara Server REST Interfaces

The Payara Server REST interfaces support the following authentication schemes for securing the REST interfaces:

- Basic authentication over a secure connection
- Authentication by using session tokens

When security is enabled, you must specify `HTTPS` as the protocol in the URLs to REST resources and provide a username and password.

Setting up Basic Authentication over a Secure Connection

Setting up basic authentication over a secure connection to secure Payara Server REST interfaces involves the following sequence of tasks:

1. Adding an `admin-realm` user to the `asadmin` user group
2. Enabling Secure Sockets Layer (SSL)

For information about how to perform these tasks from the command line, see the following documentation:

- to-create-an-authentication-realm in the Payara Server Security Guide
- to-create-a-file-user in the Payara Server Security Guide
- configure-an-http-listener-for-ssl

For information about how to perform these tasks by using the Administration Console, see the following topics in the Administration Console online help:

- To Add a User to the Admin Realm
- To Edit SSL Settings for a Protocol

To Secure REST Interfaces by Using Session Tokens

Basic authentication requires a REST client to cache a user's credentials to enable the client to pass the credentials with each request. If you require a REST client not to cache credentials, your client must use session tokens for authentication.

1. Request a session token by using the `GET` method on the resource at `http://host:port/management/sessions`.

   Payara Server uses basic authentication to authenticate the client, generates a session token, and passes the token to the client.

2. In each subsequent request that requires authentication, use the token to authenticate the client.
3. Create a cookie that is named `gfresttoken` the value of which is the token.
4. Send the cookie with the request.
5. When the token is no longer required, retire the token by using the `DELETE` method on the resource at `http://host:port/management/sessions/{tokenvalue}`.

CAUTION	If a client does not explicitly retire a token, the token is retired after 30 minutes of inactivity.

Formats for Resource Representation of Configuration Objects

The Payara Server REST interfaces represent resources for configuration objects in the following formats:

- JSON
- XML
- HTML

Payara Server enables you to specify the resource representation through the filename extension in the URL or through the HTTP header:

- To specify the resource representation through the filename extension in the URL, specify the appropriate extension as follows:
  - For JSON, specify the `.json` extension.
  - For XML, specify the `.xml` extension.
  - For HTML, omit the extension.
- How to specify the resource representation through the HTTP header depends on the client that you are using to access the resource. For example, if you are using the cURL utility, specify the resource representation through the `-H` option as follows:
  - For JSON, specify `-H "Accept: application/json"`.
  - For XML, specify `-H "Accept: application/xml"`.
  - For HTML, omit the `-H` option.

JSON Resource Representation for Configuration Objects

The general format for the JSON representation of a resource for a configuration object is as follows:

```
{
  "command":"resource",
  "exit_code":"code",
  "extraProperties": {
    "commands":["command-list"],
    "methods":["method-list"],
    "entity":"attributes",
    "childResources":"children"
  }
}
```

The replaceable items in this format are as follows:

resource: The name of the resource. **code**: The result of the attempt to get the resource. **command-list**: One or more metadata sets separated by a comma (`,`) that represent the `asadmin` subcommands for performing non—CRUD operations on the resource. For the format of each metadata set, see json-representation-of-a-command-in-a-command-list. **method-list**: One or more metadata sets separated by a comma (`,`) that represent the methods that the resource supports. For the format of each metadata set, see json-

representation-of-a-method-in-a-method-list. **attributes**: Zero or more name-value pairs separated by a comma (`,`). Each name-value pair is specified as `name:value`. **children**: Zero or more child resources separated by a comma (`,`). Each child resource is specified as `resource-name:url`.

+ resource-name;; The name of the resource as displayed in client applications that access the parent of the resource. url;; The URL to the child resource.

JSON Representation of a Command in a Command List

The JSON representation of a command in a command list is as follows:

```
{
  "path":"command-path",
  "command":"command-name",
  "method":"rest-method"
}
```

The replaceable items in this format are as follows:

command-path: The relative path to REST resource that represents the command. This path is relative to the URL of the REST resource that is the parent of the resource that represents the command. **command-name**: The name of the command as displayed in client applications that access the resource. **rest-method**: The REST resource method that the command invokes when the command is run. The method is `GET`, `POST`, or `DELETE`.

JSON Representation of a Method in a Method List

The JSON representation of a method in a method list is as follows:

```
{
    "name":"method-name",
    "messageParameters":{
        "message-parameter-list":[]
    },
    "queryParameters":{
        "query-parameter-list":[]
    }
}
```

The replaceable items in this format are as follows:

method-name: The name of the method, which is `GET`, `POST`, or `DELETE`. **message-parameter-list**: Zero or more metadata sets separated by a comma (`,`) that represent the message parameters that are allowed for the method. For the format of each metadata set, see json-representation-of-a-message-parameter-or-a-query-parameter. **query-parameter-list**: Zero or more metadata sets separated by a comma (`,`) that represent the query parameters that are allowed for the method. For the format of each metadata set, see json-representation-of-a-message-parameter-or-a-query-parameter.

JSON Representation of a Message Parameter or a Query Parameter

The JSON representation of a message parameter or a query parameter is as follows:

```
{
  "parameter-name":{
    "attr-1" : "value",
    "attr-2" : "value",
    "attr-3" : "value"
  }
}
```

The replaceable items in this format are as follows:

parameter-name: The name of the parameter. **attribute-list**: A comma-separated list of name-value pairs of attributes for the parameter. Each pair is in the following format:

+

```
{
  "name":"value"
}
```

Possible attributes are as follows:

+ `defaultValue`;; The default value of the parameter. `acceptableValues`;; The set or range of acceptable values for the parameter. `type`;; The data type of the parameter, which is one of the following types: * `boolean` * `int` * `string` `optional`;; Indicates whether the parameter is optional. `key`;; Indicates whether the parameter is a key parameter.

Example JSON Resource Representation for a Configuration Object

This example shows the JSON representation of the resource for the node `sj01`. In this example, the DAS is running on the local host and the HTTP port for administration is 4848. The URL to the resource in this example is `http://localhost:4848/management/domain/nodes/node/sj01`.


```
{
  "command":"Node",
  "exit_code":"SUCCESS",
  "extraProperties":{
    "commands":[
      {"path":"_delete-node","command":"delete-node","method":"DELETE"},
      {"path":"_update-node","command":"_update-node","method":"POST"},
      {"path":"ping-node-ssh","command":"ping-node-ssh","method":"GET"},
      {"path":"update-node-ssh","command":"update-node-ssh","method":"POST"},
      {"path":"update-node-config","command":"update-node-config","method":
"POST"}],
    "methods":[
      {"name":"GET"},
      {"name":"POST","messageParameters":{
        "installDir":{"optional":"true","type":"string","key":"false"},
        "nodeDir":{"optional":"true","type":"string","key":"false"},
        "nodeHost":{"optional":"true","type":"string","key":"false"},
        "type":{"optional":"true","type":"string","key":"false"}
        }
      }
    ],
    "entity":{
      "installDir":"\/export\/payara7",
      "name":"sj01",
      "nodeDir":null,
      "nodeHost":
      "sj01.example.com",
      "type":"SSH"
    },
    "childResources":{
      "application-ref":
       "https:\/\/localhost:4848\/management\/domain\/nodes\/node\/sj01
\/application-ref",
      "resource-ref":
       "https:\/\/localhost:4848\/management\/domain\/nodes\/node\/sj01\/resource-
ref",
      "ssh-connector":
       "https:\/\/localhost:4848\/management\/domain\/nodes\/node\/sj01\/ssh-
connector"
    }
  }
}
```


XML Resource Representation for Configuration Objects

The general format for the XML representation of a resource for a configuration object

is as follows:

```
<map>
 <entry key="extraProperties">
  <map>
   <entry key="methods">
    <list>
     methods
    </list>
   </entry>
   <entry key="entity">
    <map>
     attributes
    </map>
   </entry>
   <entry key="commands">
    <list>
     commands
    </list>
   </entry>
   <entry key="childResources">
    <map>
     children
    </map>
   </entry>
  </map>
 </entry>
 <entry key="message"></entry>
 <entry key="exit_code" value="code"></entry>
 <entry key="command" value="resource"></entry>
</map>
```

The replaceable items in this format are as follows:

methods: One or more XML elements that represent the methods that the resource supports. For the format of each element, see xml-representation-of-a-resource-method. **attributes**: Zero or more XML elements that represent the attributes of the resource. Each element specifies a name-value pair as follows:

+

```
<entry key="name" value="value"></entry>
```

commands: One or more XML elements that represent the `asadmin` subcommands for performing non—CRUD operations on the resource. For the format of each element, see xml-representation-of-a-command. **children**: Zero or more XML elements that represent the children of the resource. Each element is specified as follows:

+

```
<entry key="resource-name" value="url"></entry>
```

+ resource-name;; The name of the resource as displayed in client applications that access the parent of the resource. url;; The URL to the child resource. **code**: The result of the attempt to get the resource. **resource**: The name of the resource.

XML Representation of a Resource Method

The XML representation of a method in a method list is as follows:

```
<map>
 <entry key="name" value="method-name"></entry>
 <entry key="messageParameters">
  message-parameter-list
 </entry>
 <entry key="queryParameters">
  message-parameter-list
 </entry>
</map>
```

The replaceable items in this format are as follows:

method-name: The name of the method, which is `GET`, `POST`, or `DELETE`. **message-parameter-list**: Zero or more XML elements that represent the message parameters that are allowed for the method. For the format of each element, see xml-representation-of-a-message-parameter-or-a-query-parameter. **query-parameter-list**: Zero or more XML elements that represent the query parameters that are allowed for the method. For the format of each element, see xml-representation-of-a-message-parameter-or-a-query-parameter.

XML Representation of a Command

The XML representation of a command is as follows:

```
<map>
    <entry key="command" value="command-name"></entry>
    <entry key="path" value="command-path"></entry>
    <entry key="method" value="rest-method"></entry>
</map>
```

The replaceable items in this format are as follows:

command-name: The name of the command as displayed in client applications that access the resource. **command-path**: The relative path to REST resource that represents the command. This path is relative to the URL of the REST resource that is the parent of the resource that represents the command. **rest-method**: The REST resource method that the command invokes when the command is run. The method is `GET`, `POST`, or `DELETE`.

XML Representation of a Message Parameter or a Query Parameter

The XML representation of a message parameter or a query parameter is as follows:

```
<map>
 <entry key="parameter-name">
  <map>
   attributes
  </map>
 </entry>
</map>
```

The replaceable items in this format are as follows:

parameter-name: The name of the parameter. **attributes**: One or more XML elements that represent the attributes for the parameter. Each element specifies a name-value pair as follows:

+

```
<entry key="name" value="value"></entry>
```

+ Possible attributes are as follows:

`defaultValue`

The default value of the parameter.

`acceptablevalues`

The set or range of acceptable values for the parameter.

`type`

The data type of the parameter, which is one of the following types:

- `boolean`
- `int`
- `string`

`optional`

Indicates whether the parameter is optional. If `true`, the parameter is optional.If `false`, the parameter is required.

`key`

Indicates whether the parameter is key. If `true`, the parameter is key. If `false`, the parameter is not key.

Example XML Resource Representation

This example shows the XML representation of the resource for the node `sj01`. In this example, the DAS is running on the local host and the HTTP port for administration is `4848`. The URL to the resource in this example is `http://localhost:4848/management/domain/nodes/node/sj01`.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<map>
 <entry key="extraProperties">
  <map>
   <entry key="methods">
    <list>
     <map>
      <entry key="name" value="GET"/>
     </map>
     <map>
      <entry key="name" value="POST"/>
      <entry key="messageParameters">
       <map>
        <entry key="installDir">
         <map>
          <entry key="optional" value="true"/>
          <entry key="type" value="string"/>
          <entry key="key" value="false"/>
         </map>
        </entry>
        <entry key="nodeDir">
```

```
          <map>
           <entry key="optional" value="true"/>
           <entry key="type" value="string"/>
           <entry key="key" value="false"/>
          </map>
         </entry>
         <entry key="type">
          <map>
           <entry key="optional" value="true"/>
           <entry key="type" value="string"/>
           <entry key="key" value="false"/>
          </map>
         </entry>
         <entry key="nodeHost">
          <map>
           <entry key="optional" value="true"/>
           <entry key="type" value="string"/>
           <entry key="key" value="false"/>
          </map>
         </entry>
        </map>
       </entry>
      </map>
     </list>
    </entry>
    <entry key="entity">
     <map>
      <entry key="installDir" value="/export/payara7"/>
      <entry key="name" value="sj01"/>
      <entry key="nodeDir" value=""/>
      <entry key="type" value="SSH"/>
      <entry key="nodeHost" value="sj01example.com"/>
     </map>
    </entry>
    <entry key="commands">
     <list>
      <map>
       <entry key="command" value="delete-node"/>
       <entry key="path" value="_delete-node"/>
       <entry key="method" value="DELETE"/>
      </map>
      <map>
       <entry key="command" value="_update-node"/>
       <entry key="path" value="_update-node"/>
       <entry key="method" value="POST"/>
      </map>
      <map>
       <entry key="command" value="ping-node-ssh"/>
       <entry key="path" value="ping-node-ssh"/>
       <entry key="method" value="GET"/>
      </map>
      <map>
       <entry key="command" value="update-node-ssh"/>
       <entry key="path" value="update-node-ssh"/>
       <entry key="method" value="POST"/>
      </map>
      <map>
       <entry key="command" value="update-node-config"/>
       <entry key="path" value="update-node-config"/>
       <entry key="method" value="POST"/>
```

```
      </map>
     </list>
    </entry>
    <entry key="childResources">
     <map>
      <entry key="application-ref"
       value="https://localhost:4848/management/domain/nodes/node/sj01/application-
ref"/>
      <entry key="ssh-connector"
       value="https://localhost:4848/management/domain/nodes/node/sj01/ssh-
connector"/>
      <entry key="resource-ref"
       value="https://localhost:4848/management/domain/nodes/node/sj01/resource-
ref"/>
     </map>
    </entry>
   </map>
  </entry>
  <entry key="message"/>
  <entry key="exit_code" value="SUCCESS"/>
  <entry key="command" value="Node"/>
</map>
```

HTML Resource Representation for Configuration Objects

The format for the HTML representation of a resource for a configuration object is a web page that provides the following information about the resource:

- A list of the attributes of the resource and their values.
- A list of the methods and method parameters that the resource supports.

  Each method and its parameters are presented as a field of the appropriate type in an HTML form.

- A list of hypertext links to the children of the resource.
- A list of hypertext links to resources that represent `asadmin` subcommands for non-CRUD operations on the resource.

Formats for Resource Representation of Monitoring Objects

The Payara Server REST interfaces represent resources for monitoring data in the following formats:

- JSON
- XML

- HTML

JSON Resource Representation for Monitoring Objects

The general format for the JSON representation of a resource for a monitoring object is as follows:

```
{
  "message":"",
  "command":"Monitoring Data",
  "exit_code":"code",
  "extraProperties":{
    "entity":{
    },
    "childResources":{
    }
  }
}
```

The replaceable items in this format are as follows:

code: The result of the attempt to get the resource. **entity**: Contains the statistics that the monitoring object provides. For the format of each metadata set, see json-representation-of-a-statistic-in-a-statistics-list. **childResources**: Zero or more child resources separated by a comma (`,`). Each child resource is specified as "resource-name":"url".
resource-name;; The name of the resource as displayed in client applications that access the parent of the resource. url;; The URL to the child resource.

JSON Representation of a Statistic in a Statistics List

The JSON representation of a counter statistic in a statistics list is as follows:

```
{
  "count": "number",
  "lastsampletime": "dateTime",
  "description": "string",
  "unit": "string",
  "name": "string",
  "starttime": "number"
}
```

The JSON representation of a range statistic in a statistics list is as follows:

```
{
  "highwatermark": "number",
  "lowwatermark": "number",
  "current": "number",
  "lastsampletime": "dateTime",
  "description": "string",
  "unit": "string",
  "name": "string",
  "starttime": "number"
}
```

The replaceable items in these formats are as follows:

statistic: The name of the statistic. **count**: Counter statistics only: The current value of the statistic. **highest-value**: Range statistics only: The highest value of the statistic since monitoring of the statistic began. **lowest-value**: Range statistics only: The lowest value of the statistic since monitoring of the statistic began. **current-value**: Range statistics only: The lowest value of the statistic since monitoring of the statistic began. **last-sample-time**: The time in UNIX time at which the statistic was last sampled. **description**: A textual description of what the statistic represents. **unit**: The unit of measurement of the statistic, which is one of the following units of measurement:

+ `count`;; The cumulative value of an attribute that increases with time. `range`;; The lowest value, highest value, and current value of an attribute that can increase or decrease with time. `boundedrange`;; The lowest value, highest value, and current value of an attribute that can increase or decrease with time and has fixed limits. `string`;; A string that represents an attribute value. A string statistic is similar to a count, except that the values are not ordered. Typically, a string statistic represents the state of an object, for example, `CONNECTED`, `CLOSED`, or `DISCONNECTED`. `time`;; Values of an attribute that provide the following timing measurements for an operation:
* The number of times the operation was performed. * The maximum amount of time to perform the operation once. * The minimum amount of time to perform the operation once. * The total amount of time that has been spent performing the operation. * The average amount of time to perform the operation. **name**: The name of the statistic as displayed in client applications that access the resource that contains the statistic. **start-time**: The time in UNIX time at which monitoring of the statistic began.

Example JSON Resource Representation for a Monitoring Object

This example shows the JSON representation of the monitoring object that provides class loader statistics for the virtual machine for the Java platform. In this example, the DAS is running on the local host and the HTTP port for administration is `4848`. The URL to the resource in this example is `http://localhost:4848/monitoring/domain/server/jvm/class-loading-system`.

```
{
  "message": "",
  "command":"Monitoring Data",
  "exit_code":"SUCCESS",
  "extraProperties":{
    "entity":{
      "loadedclass-count":{
        "count":8521,
        "lastsampletime":1300726961018,
        "description":"Number of classes currently loaded in the Java virtual
machine",
        "unit":"count",
        "name":"LoadedClassCount",
        "starttime":1300483924126
      },
      "totalloadedclass-count":{
        "count":8682,
        "lastsampletime":1300726961018,
        "description":"Total number of classes that have been loaded since the Java
virtual machine has started execution",
        "unit":"count",
        "name":"TotalLoadedClassCount",
        "starttime":1300483924127
      },
      "unloadedclass-count":{
        "count":161,
        "lastsampletime":1300726961018,
        "description":"Total number of classes unloaded since the Java virtual
machine has started execution",
        "unit":"count",
        "name":"UnLoadedClassCount",
        "starttime":1300483924127
      }
    },"childResources":{}
  }
}
```

XML Resource Representation for Monitoring Objects

The general format for the XML representation of a resource for a monitoring object is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<map>
 <entry key="extraProperties">
  <map>
   <entry key="entity">
    <map>
     statistics
    </map>
   </entry>
   <entry key="childResources">
    <map>
     children
    </map>
   </entry>
  </map>
 </entry>
 <entry key="message" value=""></entry>
 <entry key="exit_code" value="code"></entry>
 <entry key="command" value="Monitoring Data"></entry>
</map>
```

The replaceable items in this format are as follows:

statistics: Zero or more XML elements that represent the statistics that the monitoring object provides. For the format of each element, see xml-representation-of-a-statistic. **children**: Zero or more XML elements that represent the children of the resource. Each element is specified as follows:

+

```
<entry key="resource-name" value="url"></entry>
```

resource-name: The name of the resource as displayed in client applications that access the parent of the resource. **url**: The URL to the child resource. **code**: The result of the attempt to get the resource.

XML Representation of a Statistic

The XML representation of a counter statistic is as follows:

```
<entry key="statistic">
    <map>
        <entry key="unit" value="unit"></entry>
        <entry key="starttime">
            <number>start-time</number>
        </entry>
```

```
        <entry key="count">
            <number>count</number>
        </entry>
        <entry key="description" value="description"></entry>
        <entry key="name" value="name"></entry>
        <entry key="lastsampletime">
            <number>last-sample-time</number>
        </entry>
    </map>
</entry>
```

The XML representation of a range statistic is as follows:

```
<entry key="statistic">
    <map>
        <entry key="unit" value="unit"></entry>
        <entry key="starttime">
            <number>start-time</number>
        </entry>
        <entry key="highwatermark">
            <number>highest-value</number>
        </entry>
        <entry key="lowwatermark">
            <number>lowest-value</number>
        </entry>
        <entry key="current">
            <number>current-value</number>
        </entry>
        <entry key="description" value="description"></entry>
        <entry key="name" value="name"></entry>
        <entry key="lastsampletime">
            <number>last-sample-time</number>
        </entry>
    </map>
</entry>
```

The replaceable items in these formats are as follows:

statistic: The name of the statistic. **unit**: The unit of measurement of the statistic, which is one of the following units of measurement:

+ `count`;; The cumulative value of an attribute that increases with time. `range`;; The lowest value, highest value, and current value of an attribute that can increase or decrease with time. `boundedrange`;; The lowest value, highest value, and current value of an attribute that can increase or decrease with time and has fixed limits. `string`;; A string that represents an attribute value. A string statistic is similar to a count, except that the values are not ordered. Typically, a string statistic represents the state of an

object, for example, `CONNECTED`, `CLOSED`, or `DISCONNECTED`. `time`;; Values of an attribute that provide the following timing measurements for an operation:

+ * The number of times the operation was performed. * The maximum amount of time to perform the operation once. * The minimum amount of time to perform the operation once. * The total amount of time that has been spent performing the operation. * The average amount of time to perform the operation. **start-time**: The in time in UNIX time at which monitoring of the statistic began. **count**: Counter statistics only: The current value of the statistic. **highest-value**: Range statistics only: The highest value of the statistic since monitoring of the statistic began. **lowest-value**: Range statistics only: The lowest value of the statistic since monitoring of the statistic began. **current-value**: Range statistics only: The lowest value of the statistic since monitoring of the statistic began. **description**: A textual description of what the statistic represents. **name**: The name of the statistic as displayed in client applications that access the resource that contains the statistic. **last-sample-time**: The time in UNIX time at which the statistic was last sampled.

Example XML Resource Representation for a Monitoring Object

This example shows the XML representation of the monitoring object that provides class loader statistics for the virtual machine for the Java platform.

In this example, the DAS is running on the local host and the HTTP port for administration is `4848`. The URL to the resource in this example is `http://localhost:4848/monitoring/domain/server/jvm/class-loading-system`.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<map>
 <entry key="extraProperties">
  <map>
   <entry key="entity">
    <map>
     <entry key="unloadedclass-count">
      <map>
       <entry key="unit" value="count"/>
       <entry key="starttime">
        <number>1300483924127</number>
       </entry><entry key="count">
        <number>161</number>
```

```
        </entry>
        <entry key="description" value="Total number of classes unloaded since the
Java virtual machine has started execution"/>
        <entry key="name" value="UnLoadedClassCount"/>
        <entry key="lastsampletime">
         <number>1300726989505</number>
        </entry>
       </map>
      </entry>
      <entry key="totalloadedclass-count">
       <map>
        <entry key="unit" value="count"/>
        <entry key="starttime">
         <number>1300483924127</number>
        </entry>
        <entry key="count">
          <number>8682</number>
        </entry>
        <entry key="description" value="Total number of classes that have been
         loaded since the Java virtual machine has started execution"/>
        <entry key="name" value="TotalLoadedClassCount"/>
        <entry key="lastsampletime">
         <number>1300726989505</number>
        </entry>
       </map>
     </entry>
      <entry key="loadedclass-count">
       <map>
        <entry key="unit" value="count"/>
        <entry key="starttime">
         <number>1300483924126</number>
        </entry><entry key="count">
         <number>8521</number>
        </entry>
        <entry key="description" value="Number of classes currently loaded in
         the Java virtual machine"/>
        <entry key="name" value="LoadedClassCount"/>
        <entry key="lastsampletime">
         <number>1300726989505</number>
        </entry>
       </map>
      </entry>
     </map>
    </entry>
    <entry key="childResources">
     <map/>
    </entry>
   </map>
  </entry>
  <entry key="message" value=""/>
  <entry key="exit_code" value="SUCCESS"/>
  <entry key="command" value="Monitoring Data"/>
</map>
```

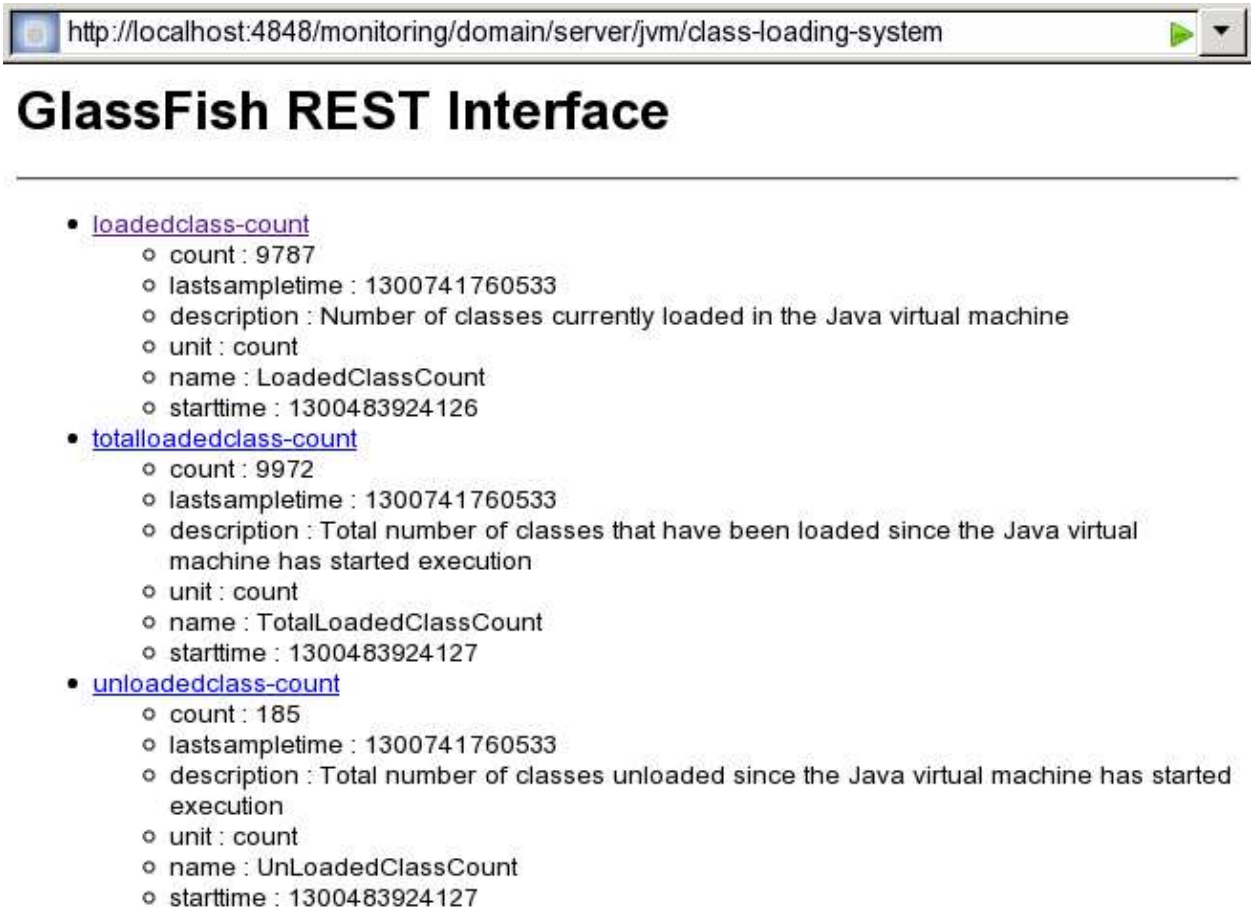
HTML Resource Representation for Monitoring Objects

The format for the HTML representation of a resource for a monitoring object is a web page that provides the following information about the resource:

- A list of the statistics that the resource provides.
- A list of hypertext links to the children of the resource.

The following figure shows the web page for the REST resource that provides class loader statistics for the virtual machine for the Java platform.

Web Page for the REST Resource That Provides Class Loader Statistics



Formats for Resource Representation of Log File Details

The Payara Server REST interfaces represent resources for log file details in the following formats:

- JSON
- XML

JSON Resource Representation for Log File Details

The general format for the JSON representation of a resource for log file details is as follows:

```
{
  "records": [
  ]
}
```

The replaceable item in this format is a record list array, which is one or more metadata sets separated by a comma (`,`) that represent the log records in the log file. For the format of each metadata set, see json-representation-of-a-log-record-in-a-record-list.

JSON Representation of a Log Record in a Record List

The JSON representation of a log record in a record list is as follows:

```
{
  "recordNumber":"record-number",
  "loggedDateTimeInMS":"logged-date",
  "loggedLevel":"log-level",
  "productName":"product-name",
  "loggerName":"logger-class-name",
  "nameValuePairs":"_ThreadID=thread-id;_ThreadName=thread-name;",
  "messageID":"message-id",
  "Message":"message-text"
}
```

The replaceable items in this format are as follows:

record-number: A serial number in the form of a decimal integer that uniquely identifies the log record. **logged-date**: The date and time in UNIX time that the record was created. **log-level**: The severity level of the message in the log record. For more information, see setting-log-levels. **product-name**: The current name and version of the server runtime. **logger-class-name**: The fully qualified name of the Java class of the logger class that created the log message. Each component of Payara Server provides its own logger class.

+ For detailed information about the names of logger classes in the Payara Server, see logger-namespaces. **thread-id**: The numerical identifier of the thread that created the

message. **thread-name**: The name of the thread that created the message. **message-id**: A unique identifier for the message. For messages from Payara Server, this identifier consists of a module code and a numerical value, for example, `CORE5004`. All `SEVERE` and `WARNING` messages and some `INFO` messages from Payara Server contain a message identifier. **message-text**: The text of the log message.

Example JSON Resource Representation for Log File Details

This example shows the JSON representation of the resource for log file details. In this example, the DAS is running on the local host and the HTTP port for administration is `4848`. The URL to the resource in this example is `http://localhost:4848/management/domain/view-log/details`.


```
{
  "records": [
    {
      "recordNumber":475,
      "loggedDateTimeInMS":1300743782815,
      "loggedLevel":"INFO",
      "productName":"Payara Server 7.2.0",
      "loggerName":"org.glassfish.admingui",
      "nameValuePairs": "_ThreadID=25;_ThreadName=Thread-1;",
      "messageID":"",
      "Message":"Admin Console: Initializing Session Attributes..."
    },
    {
      "recordNumber":474,
      "loggedDateTimeInMS":1300728893368,
      "loggedLevel":"INFO",
      "productName":"Payara 7.2.0",
      "loggerName"
:"java.enterprise.system.core.com.sun.enterprise.v3.admin.adapter",
      "nameValuePairs":"_ThreadID=238;_ThreadName=Thread-1;",
      "messageID":"",
      "Message":"The Admin Console application is loaded."
    },
    {
      "recordNumber":473,
      "loggedDateTimeInMS":1300728893367,
      "loggedLevel":"INFO",
      "productName":"Payara 7.2.0",
      "loggerName":"jakarta.enterprise.system.core.com.sun.enterprise.v3.server",
      "nameValuePairs":"_ThreadID=238;_ThreadName=Thread-1;",
      "messageID":"CORE10010",
      "Message":"Loading application __admingui done in 40,063 ms"
    }
  ]
}
```

XML Resource Representation for Log File Details

The general format for the XML representation of a resource for log file details is as follows:

```
<records>
 records
</records>
```

The replaceable item in this format is the records, which is one or more XML elements that represent the log records in the log file. For the format of each element, see xml-representation-of-a-log-record.

XML Representation of a Log Record

The XML representation of a log record is as follows:

```
<record loggedDateTimeInMS="logged-date" loggedLevel="log-level"
loggerName="logger-class-name" messageID="message-id"
nameValuePairs="_ThreadID=thread-id;_thread-name;" productName="product-name"
recordNumber="record-number"/>
```

The replaceable items in this format are as follows:

logged-date: The date and time in UNIX time that the record was created. **log-level**: The severity level of the message in the log record. For more information, see setting-log-levels. **logger-class-name**: The fully qualified name of the Java class of the logger class that created the log message. Each component of Payara Server provides its own logger class.

+ For detailed information about the names of logger classes in Payara Server, see logger-namespaces. **message-id**: A unique identifier for the message. For messages from Payara Server, this identifier consists of a module code and a numerical value, for example, `CORE5004`. All `SEVERE` and `WARNING` messages and some `INFO` messages from Payara Server contain a message identifier. **thread-id**: The numerical identifier of the thread that created the message. **thread-name**: The name of the thread that created the message. **product-name**: The current name and version of the server runtime. **record-number**: A serial number in the form of a decimal integer that uniquely identifies

the log record.

Example XML Resource Representation for Log File Details

This example shows the XML representation of the resource for log file details. In this example, the DAS is running on the local host and the HTTP port for administration is 4848. The URL to the resource in this example is `http://localhost:4848/management/domain/view-log/details`.

```
<records>
 <record loggedDateTimeInMS="1300743782815" loggedLevel="INFO"
loggerName="org.glassfish.admingui" messageID=""
nameValuePairs="_ThreadID=25;_ThreadName=Thread-1;" productName="Payara 7.2.0"
recordNumber="475"/>
 <record loggedDateTimeInMS="1300728893368" loggedLevel="INFO"
loggerName="jakarta.enterprise.system.core.com.sun.enterprise.v3.admin.adapter"
messageID="" nameValuePairs="_ThreadID=238;_ThreadName=Thread-1;"
productName="Payara 7.2.0" recordNumber="474"/>
 <record loggedDateTimeInMS="1300728893367" loggedLevel="INFO"
loggerName="jakarta.enterprise.system.core.com.sun.enterprise.v3.server"
messageid="core10010" nameValuePairs="_ThreadID=238;_ThreadName=Thread-1;"
productName="Payara 7.2.0" recordNumber="473"/>
</records>
```

Supported Content Types in Requests to REST Resources

The Payara Server REST interfaces support the following types in the content-type header of a client request:

- JSON
- XML
- Form URL encoded

How to specify the type in the content-type header depends on how you are sending the request. For example, if you are using the cURL utility, specify the type through the `-H` option as follows:

- For JSON, specify `-H "Content-type: application/json"`.
- For XML, specify `-H "Content-type: application/xml"`.
- For form URL encoded, specify `-H "Content-type: application/x-www-form-urlencoded"`.

Administering Domains

This chapter provides procedures for administering domains in the Payara Server environment by using the `asadmin`command-line utility.

Instructions for accomplishing the tasks in this chapter by using the Administration Console are contained in the Administration Console online help.

About Administering Domains

A domain contains a group of Payara Server instances that are administered together. Each domain has a domain administration server (DAS) that hosts administrative applications.

Payara Server Instances

A Payara Server instance is a single Virtual Machine for the Java platform (Java Virtual Machine or JVM machine) on a single node in which Payara Server is running. A node defines the host where the Payara Server instance resides. The JVM machine must be compatible with the Jakarta EE.

Payara Server instances form the basis of an application deployment.

Whenever a domain is created, Payara Server creates a default instance that is named `server`. If a single instance meets your requirements, you can use this instance for deploying applications without the need to administer Payara Server instances explicitly. You administer the default instance when you administer its domain.

If you require multiple instances, you must administer the instances explicitly. For more information, see "administering-payara-server-instances" in the Payara Server High Availability section.

For an instance, you can also create virtual servers. Virtual servers do not span instances. For many purposes, you can use virtual servers instead of multiple instances in operational deployments. Virtual servers enable you to offer, within a single instance, separate domain names, IP addresses, and some administration capabilities to organizations or individuals. To these users, a virtual server behaves like a dedicated

web server, but without the hardware and basic web server maintenance.

For more information about virtual servers, see administering-virtual-servers.

Domains for Administering Payara Server

A domain is an administrative boundary that contains a group of Payara Server instances that are administered together. Each instance can belong to only one domain. A domain provides a preconfigured runtime for user applications. Each domain has its own configuration data, log files, and application deployment areas that are independent of other domains. If the configuration is changed for one domain, the configurations of other domains are not affected.

Domains enable different organizations and administrators to share securely a single Payara Server installation. Each organization or administrator can administer the instances in a single domain without affecting the instances in other domains.

At installation time, Payara Server creates a default domain that is named `domain1`. After installation, you can create additional domains as necessary.

When a domain is created, you are prompted for the administration username and password. If you accept the default, the user `admin` is created without password. To reset the administration password, see to-change-an-administration-password in Payara Server Security Guide.

Domain Administration Server (DAS)

The domain administration server (DAS) is a specially designated Payara Server instance that hosts administrative applications. The DAS is similar to any other Payara Server instance, except that the DAS has additional administration capabilities. The DAS authenticates the administrator, accepts requests from administration tools, and communicates with other instances in the domain to carry out the requests from administration tools.

Each domain has its own DAS with a unique administration port number. The default administration port is 4848, but a different port can be specified when a domain is

created.

The DAS has the master copy of the configuration data for all instances in a domain. If an instance is destroyed, for example, because a host failed, the instance can be re-created from the data in the DAS.

The DAS is the default Payara Server instance in a domain and is named `server`. If a single instance meets your requirements, you can use the DAS for deploying applications and for administering the domain.

The graphical Administration Console communicates with a specific DAS to administer the domain that is associated with the DAS. Each Administration Console session enables you to configure and manage only one domain. If you create multiple domains, you must start a separate Administration Console session to manage each domain.

Creating, Logging in to, and Deleting a Domain

To Create a Domain

After installing Payara Server and creating the default domain (`domain1`), you can create additional domains by using the local `create-domain` subcommand. This subcommand creates the configuration of a domain. Any user who has access to the `asadmin` utility on a given system can create a domain and store the domain configuration in a folder of choice. By default, the domain configuration is created in the default directory for domains. You can override this location to store the configuration elsewhere.

You are required to specify an administrative user when you create a domain, or you can accept the default login identity which is username `admin` with no password.

1. Select a name for the domain that you are creating.
   You can verify that a name is not already in use by using the "`list-domains`" subcommand
2. Create a domain by using the "`create-domain`" subcommand.
   Information about the options for this subcommand is included in this help page.

3. Type an admin user name and password for the domain.

   To avoid setting up an admin login, you can accept the default `admin`, with no password. Pressing Return also selects the default.

Example 3-1 Creating a Domain

This example creates a domain named `domain1` . When you type the command, you might be prompted for login information.

```
asadmin> create-domain --adminport 4848 domain1
Enter admin user name[Enter to accept default]>
Using port 4848 for Admin.
Default port 8080 for HTTP Instance is in use. Using 1161
Using default port 7676 for JMS.
Using default port 3700 for IIOP.
Using default port 8081 for HTTP_SSL.
Using default port 3820 for IIOP_SSL.
Using default port 3920 for IIOP_MUTUALAUTH.
Default port 8686 for JMX_ADMIN is in use. Using 1162
Distinguished Name of the self-signed X.509 Server Certificate is:
[CN=moonbeam.gateway.2wire.net,OU=Payara,O=Payar Foundation,L=Great Malvern,ST
=Worcestershire,C=UK]]
Domain domain1 created.
Command create-domain executed successfully.
```

To start the Administration Console in a browser, enter the URL in the following format:

```
http://hostname:4848
```

For this example, the domain's log files, configuration files, anddeployed applications now reside in the following directory:

```
domain-root-dir`/mydomain`
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-domain` at the command line.

To Create a Domain from a Custom Template

A custom template enables you to customize the configuration of any domain that you create from the template.

1. Create a domain to use as the basis for the template.
   For more information, see to-create-a-domain.
2. Use the `asadmin` utility or the Administration Console to configure the domain.
   Your configuration changes will be included in the template that you create from the domain.
3. Copy the domain's `domain.xml` file under a new name to the as-install`/lib/templates` directory.
   A domain's `domain.xml` file is located in the domain-dir`/config` directory.
4. In a plain text editor, edit the file that you copied to replace with tokens values that are to be substituted when a domain is created.
   Each token is identified as `token-name` where token-name is one of the following names:
   - **`ADMIN_PORT`**: Represents the port number of the HTTP port or the HTTPS port for administration. This token is replaced with one of the following values in the command to create a domain from the template: + * The value of the `--adminport` option * The value of the `domain.adminPort` property
   - **`CONFIG_MODEL_NAME`**: Represents the name of the configuration that is created for the domain that is being created. This token is replaced with the string `server-config`.
   - **`DOMAIN_NAME`**: Represents the name of the domain that is being created. This token is replaced with the operand of `create-domain` subcommand.
   - **`HOST_NAME`**: Represents the name of the host on which the domain is being created. This token is replaced with the fully qualified host name of the host where the domain is being created.
   - **`HTTP_PORT`**: Represents the port number of the port that is used to listen for HTTP requests. This token is replaced with one of the following values in the command to create a domain from the template: + * The value of the `--instanceport` option * A value that the `create-domain` subcommand calculates from the value of the `--portbase` option * The value of the

`domain.instancePort` property

- **`HTTP_SSL_PORT`**: Represents the port number of the port that is used to listen for secure HTTP requests. This token is replaced with one of the following values in the command to create a domain from the template: + * A value that the `create-domain` subcommand calculates from the value of the `--portbase` option * The value of the `http.ssl.port` property
- **`JAVA_DEBUGGER_PORT`**: Represents the port number of the port that is used for connections to the Java Platform Debugger Architecture (JPDA) debugger. This token is replaced with one of the following values in the command to create a domain from the template: + * A value that the `create-domain` subcommand calculates from the value of the `--portbase` option * The value of the `java.debugger.port` property
- **`JMS_PROVIDER_PORT`**: Represents the port number for the Java Message Service provider. This token is replaced with one of the following values in the command to create a domain from the template: + * A value that the `create-domain` subcommand calculates from the value of the `--portbase` option * The value of the `jms.port` property
- **`JMX_SYSTEM_CONNECTOR_PORT`**: Represents the port number on which the JMX connector listens. This token is replaced with one of the following values in the command to create a domain from the template: + * A value that the `create-domain` subcommand calculates from the value of the `--portbase` option * The value of the `domain.jmxPort` property
- **`ORB_LISTENER_PORT`**: Represents the port number of the port that is used for IIOP connections. This token is replaced with one of the following values in the command to create a domain from the template: + * A value that the `create-domain` subcommand calculates from the value of the `--portbase` option * The value of the `orb.listener.port` property
- **`ORB_MUTUALAUTH_PORT`**: Represents the port number of the port that is used for secure IIOP connections with client authentication. This token is replaced with one

of the following values in the command to create a domain from the template: + * A value that the `create-domain` subcommand calculates from the value of the `--portbase` option * The value of the `orb.mutualauth.port` property

- **`ORB_SSL_PORT`**: Represents the port number of the port that is used for secure IIOP connections. This token is replaced with one of the following values in the command to create a domain from the template: + * A value that the `create-domain` subcommand calculates from the value of the `--portbase` option * The value of the `orb.ssl.port` property
- **`OSGI_SHELL_TELNET_PORT`**: Represents the port number of the port that is used for connections to the Apache Felix Remote Shell . This shell uses the Felix shell service to interact with the OSGi module management subsystem. This token is replaced with one of the following values in the command to create a domain from the template: + * A value that the `create-domain` subcommand calculates from the value of the `--portbase` option * The value of the `osgi.shell.telnet.port` property
- **`SERVER_ID`**: Represents the name of the DAS for the domain that is being created. This token is replaced with the string `server`.

5. Create the domain that you want to be based on a custom template.
   In the command to create the domain, pass the name of file that you edited in the previous step as the `--template` option of the "`create-domain`" subcommand.
6. Before starting the domain, verify that the domain's `domain.xml` file is valid.
   Use the "`verify-domain-xml`" subcommand for this purpose.
   Information about the options for this subcommand is included in the subcommand's help page.

TIP	For information about how these tokens are used in the default template, examine the `as-install/lib/templates/domain.xml` file.

See Also

- to-create-a-domain
- "`create-domain`"
- "`verify-domain-xml`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line.

- `asadmin help create-domain`
- `asadmin help verify-domain-xml`

To List Domains

Use the `list-domains` subcommand to display a list of domains and their statuses. If the domain directory is not specified, the contents of the domain-root-dir, the default for which is as-install`/domains`, is listed. If there is more than one domain, the domain name must be specified.

To list domains that were created in other directories, specify the `--domaindir` option.

List domains by using the "`list-domains`" subcommand.

Example 3-2 Listing Domains

This example lists the domains in the default domain root directory:

```
asadmin> list-domains
Name: domain1 Status: Running
Name: domain4 Status: Not Running
Name: domain6 Status: Not Running
Command list-domains executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-domain` at the command line.

To Log in to a Domain

All remote subcommands require that credentials be specified in terms of an administration user name and its password. By default, the domain is created with an identity that allows an `asadmin` user to perform administrative operations when no

identity is explicitly or implicitly specified.

The default identity is in the form of a user whose name is `admin` and has no password. If you specify no user name on the command line or on prompt, and specify no password in the `--passwordfile` option or on prompt, and you have never logged in to a domain using either the `login` subcommand or the `create-domain` subcommand with the `--savelogin` option, then the `asadmin` utility will attempt to perform a given administrative operation without specifying any identity.

A server (domain) allows administrative operations to be run using this default identity if the following conditions are true:

- The server (domain) uses file realm for authentication of administrative users.
  If this condition is not true, you will need to specify the user name and password.
- The file realm has one and only one user (what the user name is does not matter).
  If this condition is not true, you will also need to specify the user name.
- That one user has no password.
  If this condition is not true, you will need to specify the password.

By default, all of these conditions are true, unless you have created the domain with a specific user name and password. Thus, by default, the only administrative user is `admin` with no password.

Use the `login` subcommand in local mode to authenticate yourself (log in to) a specific domain. After such login, you do not need to specify the administration user or password for subsequent operations on the domain. The `login` subcommand can only be used to specify the administration password. For other passwords that remote subcommands require, use the `--passwordfile` option, or specify the password at the command prompt. You are always prompted for the administration user name and password.

There is no logout subcommand. If you want to log in to another domain, invoke `asadmin login` with new values for `--host` and `--port`.

1. Determine the name of the domain that you are logging in to.

   To list the existing domains:

```
asadmin list-domains
```

2. Log in to the domain by using the domain-subcommands command.

Example 3-3 Logging in to a Domain on a Remote Machine

This example logs into a domain located on another machine. Options are specified before the `login` subcommand.

```
asadmin> --host foo --port 8282 login
Please enter the admin user name>admin Please enter the admin password>
Trying to authenticate for administration of server at host [foo] and port [8282]
...
Login information relevant to admin user name [admin]
for host [foo] and admin port [8282] stored at [/.asadminpass] successfully.
Make sure that this file remains protected. Information stored in this
file will be used by asadmin commands to manage associated domain.
```

Example 3-4 Logging in to a Domain on the Default Port of Localhost

This example logs into a domain on `myhost` on the default port. Options are specified before the login subcommand.

```
asadmin> --host myhost login
Please enter the admin user name>admin
Please enter the admin password>
Trying to authenticate for administration of server at host [myhost] and port
[4848] ...
An entry for login exists for host [myhost] and port [4848], probably from
an earlier login operation.
Do you want to overwrite this entry (y/n)?y
Login information relevant to admin user name [admin] for host [myhost]
and admin port [4848] stored at [/home/joe/.asadminpass] successfully.
Make sure that this file remains protected. Information stored in this file will be
used by
asadmin commands to manage associated domain.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help login` at the command line. For additional information about passwords, see administering-passwords in Payara Server Security Guide.

To Delete a Domain

Use the `delete-domain` subcommand to delete an existing domain from a server. Only the root user or the operating system user who is authorized to administer the domain can run this subcommand.

Before You Begin

A domain must be stopped before it can be deleted.

1. List domains by using the "`list-domains`" subcommand.
2. If necessary, notify domain users that the domain is being deleted.
3. Ensure that the domain you want to delete is stopped.
   If needed, see to-stop-a-domain.
4. Delete the domain by using the "`delete-domain`" subcommand.

Example 3-5 Deleting a Domain

This example deletes a domain named `domain1` from the location specified.

```
asadmin> delete-domain --domaindir ..\domains domain1
Domain domain1 deleted.
Command delete-domain executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-domain` at the command line.

Starting and Stopping a Domain

To Start a Domain

When you start a domain or server, the domain administration server (DAS) is started. After startup, the DAS runs constantly, listening for and accepting requests.

If the domain directory is not specified, the domain in the default domain root directory is started. If there are two or more domains, the `domain_name` operand must be specified. Each domain must be started separately.

This subcommand is supported in local mode only.

Start a domain by using the "`start-domain`" subcommand.

Example 3-6 Starting a Domain

This example starts `domain2` in the default domain directory.

```
asadmin> start-domain domain2
```

If there is only one domain, you can omit the domain name. If you do not include the password, you might be prompted to supply it.

```
Name of the domain started: [domain1] and its location:
[C:\prelude\v3_prelude_release\distributions\web\target\glassfish
domains\domain1].
Admin port for the domain: [4848].
```

You can also view the full syntax and options of the subcommand by typing `asadmin help start-domain` at the command line.

To Stop a Domain

Stopping a domain or server shuts down its domain administration server (DAS). When stopping a domain, the DAS stops accepting new connections and then waits for all outstanding connections to complete. This shutdown process takes a few seconds. While the domain is stopped, the Administration Console and most of the `asadmin` subcommands cannot be used. This subcommand is particularly useful in stopping a runaway server. For more controlled situations, you can use the "`restart-domain`" subcommand.

1. If necessary, notify users that you are going to stop the domain.
2. Stop the domain by using the "`stop-domain`" subcommand.

Example 3-7 Stopping a Domain (or Server)

This example stops `domain1` in the default directory, where `domain1` is the only domain present in the directory.

```
asadmin> stop-domain
Waiting for the domain to stop ..........
```

```
Command stop-domain executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help stop-domain` at the command line.

To Restart a Domain

Use the `restart-domain` subcommand in remote mode to restart the Domain Administration Server (DAS) of the specified host. When restarting a domain, the DAS stops accepting new connections and then waits for all outstanding connections to complete. This shutdown process takes a few seconds. Until the domain has restarted, the Administration Console and most of the `asadmin` subcommands cannot be used.

This subcommand is particularly useful for environments where the server machine is secured and difficult to get to. With the right credentials, you can restart the server from a remote location as well as from the same machine.

If the server will not restart, use the "`stop-domain`" subcommand followed by the "`start-domain`" subcommand.

1. Ensure that the server is running.
   Remote subcommands require a running server.
2. Restart the domain by using the "`restart-domain`" subcommand.

Example 3-8 Restarting a Domain (or Server)

This example restarts `mydoimain4` in the default directory.

```
asadmin> restart-domain mydomain4
Waiting for the domain to restart ..........
Command restart-domain executed successfully.
```

Example 3-9 Restarting a Domain in a Browser

This example invokes the `restart-domain` subcommand in a browser.

```
http://yourhost:4848/__asadmin/restart-domain
```

You can also view the full syntax and options of the subcommand by typing `asadmin help restart-domain` at the command line.

Configuring a DAS or a Payara Server Instance for Automatic Restart

Use the `create-service` subcommand in local mode to configure your system to automatically restart a domain administration server (DAS) or a Payara Server instance. Payara Server enables you to configure a DAS or an instance for automatic restart on the following operating systems:

- Windows
- Linux
- Oracle Solaris

To ensure that automatic restart functions correctly on Windows, you must prevent service shutdown when a user logs out.

To Configure a DAS or an Instance for Automatic Restart on Windows

On Windows systems, the `create-service` subcommand creates a Windows service to represent the DAS or instance. The service is created in the disabled state. After this subcommand creates the service, you must use the Windows Services Manager or the Windows Services Wrapper to start, stop, uninstall, or install the service. To administer the service from the Windows command line, use the `sc.exe` tool.

This subcommand must be run as the OS-level administrator user.

1. Create the service by using the "`create-service`" subcommand.
2. After the service is created, start the service by using the Windows Services Manager or the Windows Services Wrapper.
   For example, to start the service for the default domain by using the `sc.exe` tool, type:

   ```
   C:\> sc start domain1
   ```

If you are using the `sc.exe` tool to administer the service, use the tool as follows:

- To obtain information about the service, use the `sc query` command.
- To stop the service, use the `sc stop` command.
- To uninstall the service, use the `sc delete` command.

Example 3-10 Creating a Service to Restart a DAS Automatically on Windows

This example creates a service for the default domain on a system that is running Windows.

```
asadmin> create-service
Found the Windows Service and successfully uninstalled it.
The Windows Service was created successfully.  It is ready to be started.  Here are
the details:
ID of the service: domain1
Display Name of the service:domain1 GlassFish Server
Domain Directory: C:\glassfishv3\glassfish\domains\domain1
Configuration file for Windows Services Wrapper: C:\glassfishv3\glassfish\domains\
domain1\bin\domain1Service.xml
The service can be controlled using the Windows Services Manager or you can use the
Windows Services Wrapper instead:
Start Command:  C:\glassfishv3\glassfish\domains\domain1\bin\domain1Service.exe
start
Stop Command:   C:\glassfishv3\glassfish\domains\domain1\bin\domain1Service.exe
stop
Uninstall Command:  C:\glassfishv3\glassfish\domains\domain1\bin\domain1Service.exe
uninstall
Install Command:  C:\glassfishv3\glassfish\domains\domain1\bin\domain1Service.exe
install

This message is also available in a file named PlatformServices.log in the domain's
root directory
Command create-service executed successfully.
```

Example 3-11 Querying the Service to Restart a DAS Automatically on Windows

This obtains information about the service for the default domain on a system that is running Windows.

```
C:\> sc query domain1

SERVICE_NAME: domain1
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 1  STOPPED
        WIN32_EXIT_CODE    : 1077  (0x435)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
```

```
        WAIT_HINT          : 0x0
```

To Configure a DAS or an Instance for Automatic Restart on Linux

On Linux systems, the `create-service` subcommand creates a System-V-style initialization script `/etc/init.d/GlassFish_`domain-or-instance-name and installs a link to this script in any `/etc/rc`N.d` directory that is present, where N is `0`, `1`, `2`, `3`, `4`, `5`, `6`, and `S`. After this subcommand creates the script, you must use this script to start, stop, or restart the domain or instance.

The script automatically restarts the domain or instance only during a reboot. If the domain or instance is stopped, but the host remains running, the domain or instance is not restarted automatically. To restart the domain or instance, you must run the script manually.

You might no longer require the domain or instance to be automatically restarted during a reboot. In this situation, use the operating system to delete the initialization script and the link to the script that the `create-service` subcommand creates.

The `create-service` subcommand must be run as the OS-level root user.

Create the service by using the "`create-service`" subcommand.

Example 3-12 Creating a Service to Restart a DAS Automatically on Linux

This example creates a service for the default domain on a system that is running Linux.

```
asadmin> create-service
Found the Linux Service and successfully uninstalled it.
The Service was created successfully. Here are the details:
Name of the service:domain1
Type of the service:Domain
Configuration location of the service:/etc/init.d/GlassFish_domain1
User account that will run the service: root
You have created the service but you need to start it yourself.
Here are the most typical Linux commands of interest:

* /etc/init.d/GlassFish_domain1 start
* /etc/init.d/GlassFish_domain1 stop
* /etc/init.d/GlassFish_domain1 restart

For your convenience this message has also been saved to this file:
/export/payara7/glassfish/domains/domain1/PlatformServices.log
```

```
Command create-service executed successfully.
```

To Configure a DAS or an Instance for Automatic Restart on Oracle Solaris

On Oracle Solaris systems, the `create-service` subcommand creates an Oracle Solaris Service Management Facility (SMF) service that restarts a DAS or an instance. The service grants to the process the privileges of the user that runs the process. When you create an SMF service, the default user is the superuser. If you require a different user to run the process, specify the user in `method_credential`.

If your process is to bind to a privileged port of Oracle Solaris, the process requires the `net_privaddr` privilege. The privileged ports of the Oracle Solaris operating system have port numbers less than 1024.

To determine if a user has the `net_privaddr` privilege, log in as that user and type the command `ppriv -l | grep net_privaddr`.

After you create and enable the SMF service, if the domain or instance is stopped, SMF restarts it.

Before You Begin

To run the `create-service` subcommand, you must have `solaris.smf.*` authorization. For information about how to set the authorizations, see the `useradd` man page and the `usermod` man page. You must also have write permission in the directory tree: `/var/svc/manifest/application/SUNWappserver`. Usually, the superuser has both of these permissions. Additionally, Oracle Solaris administration commands such as `svccfg`, `svcs`, and `auths` must be available in the PATH.

If a particular Payara Server domain or instance should not have default user privileges, modify the manifest of the service and reimport the service.

1. Create the service by using the "`create-service`" subcommand.
2. After the service is created, enable the service by using the `svacdm enable` command.

For example, to enable the SMF service for the default domain, type:

```
svacdm enable /appserver/domains/domain1
```

Example 3-13 Creating a Service to Restart a Domain Automatically on Oracle Solaris

This example creates a service for the default domain on a system that is running Oracle Solaris.

```
asadmin> create-service
The Service was created successfully. Here are the details:
Name of the service:application/GlassFish/domain1
Type of the service:Domain
Configuration location of the service:/home/gfuser/glassfish-installations
/glassfishv3/glassfish/domains
Manifest file location on the system:/var/svc/manifest/application
/GlassFish/domain1_home_gfuser_glassfish-installations_glassfishv3
_payara_domains/Domain-service-smf.xml.
You have created the service but you need to start it yourself.
Here are the most typical Solaris commands of interest:
* /usr/bin/svcs -a | grep domain1 // status
* /usr/sbin/svcadm enable domain1 // start
* /usr/sbin/svcadm disable domain1 // stop
* /usr/sbin/svccfg delete domain1 // uninstall
Command create-service executed successfully
```

For information about administering the service, see the following Oracle Solaris documentation:

- Managing Services (Overview) in System Administration Guide: Basic Administration
- Managing Services (Tasks) in System Administration Guide: Basic Administration
- `auths`
- `svcs`
- `svcadm`
- `svccfg`
- `useradd`
- `usermod`
- `rbac`

- `smf_security`

To Prevent Service Shutdown When a User Logs out on Windows

By default, the Java Virtual Machine (VM) receives signals from Windows that indicate that Windows is shutting down, or that a user is logging out of Windows, which causes the system to shut itself down cleanly. This behavior causes the Payara Server service to shut down. To prevent the service from shutting down when a user logs out, you must set the `-Xrs` Java VM option.

1. Ensure that the DAS is running.
2. Set the `-Xrs` Java VM option for the DAS.
   Use the "`create-jvm-options`" subcommand for this purpose.

```
asadmin> create-jvm-options -Xrs
```

3. Set the `-Xrs` Java VM option for the Java VM within which the `asadmin` utility runs. To set this option, edit the `asadmin.bat` file to add the `-Xrs` option to the line that runs the `admin-cli.jar` file.
4. In the as-install`\bin\asadmin.bat` file, edit the line to read as follows:

```
%JAVA% -Xrs -jar "%~dp0..\modules\admin-cli.jar" %*
```

5. In the as-install-parent`\bin\asadmin.bat` file, edit the line to read as follows:

```
%JAVA% -Xrs -jar "%~dp0..\glassfish\modules\admin-cli.jar" %*
```

6. If the Payara Server service is running, restart the service for your changes to take effect.

Backing up and Restoring a Domain

To Back up a Domain

Use the `backup-domain` subcommand in local mode to make a backup of a specified domain.

When you use the `backup-domain` subcommand, Payara Server creates a ZIP file backup of all the files and subdirectories in the domain's directory, domain-root-dir`/`domain-dir, except for the `backups` subdirectory.

The `backup-domain` subcommand provides several options to meet particular needs, including:

- `--backupdir` to specify a directory in which to store the backup instead of the default domain-root-dir`/`domain-dir`/backups`.
- `--description` to provide a description of the backup to be stored in the backup itself.

1. Ensure that the domain is stopped .
   The `backup-domain` subcommand operates only when the domain is stopped.
2. Back up the domain by using the "`backup-domain`" subcommand.
3. Restore the domain to its previous state, if necessary.
   Start or resume the domain.

Example 3-14 Backing up the Default Domain

This example makes a backup of the default domain, `domain1`, storing the backup file in `/net/backups.example.com/glassfish`:

```
asadmin> backup-domain --backupdir /net/backups.example.com/domains domain1
Backed up domain1 at Mon Jan 17 08:16:22 PST 2011.
Command backup-domain executed successfully
```

You can also view the full syntax and options of the subcommand by typing `asadmin help backup-domain` at the command line.

To Restore a Domain

Use the `restore-domain` subcommand in local mode to use a backup file to restore the files and subdirectories in a specified domain's directory.

The `restore-domain` subcommand can use backup files created by the `backup-domain` subcommand and by automatic backup configurations, both full backups and

configuration-only backups. Automatic backup configurations are available only in Payara Server.

1. If necessary, notify domain users that the domain is being restored from backup.
2. Ensure that the domain is stopped.
   The `restore-domain` subcommand operates only when the domain is stopped.
   To determine whether the domain is running, use the "`list-domains`" subcommand, as described in to-list-domains.
   To stop the domain, use the "`stop-domain`" subcommand as described in to-stop-a-domain.
3. Restore backup files for a domain by using the "`restore-domain`" subcommand.
4. Verify that the restore has succeeded.
5. If necessary, notify users that the domain has been restored and is available.

Example 3-15 Restoring the Default Domain

This example restores files for the default domain, `domain1`, from the most recent backup stored in a specified backup directory:

```
asadmin> restore-domain --backupdir /net/backups.example.com/domains domain1
Restored the domain (domain1) to /home/user1/payara/glassfish/domains/domain1
Command restore-domain executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin restore-domain --help` at the command line.

To List Domain Backups

Use the `list-backups` subcommand in local mode to display information about backups of a specified domain stored in a specified backup directory.

The `list-backups` subcommand provides several options to meet particular needs, including `--backupdir` to specify a directory where backups are stored instead of the default domain-dir`/backups`.

List backups by using the `list-backups` subcommand.

Example 3-16 Listing Backups of the Default Domain

This example lists the backups of the default domain, `domain1`, that are stored in the `/net/backups.example.com/glassfish` directory:

```
asadmin> list-backups --backupdir /net/backups.example.com/domains domain1
CONFIG        USER   BACKUP DATE                    FILENAME
              user1  Mon Jan 17 08:16:22 PST 2011  domain1_2011_01_17_v00001.zip
monthly-full  user1  Wed Dec 01 00:00:00 PST 2010  domain1_2010_12_01_v00001.zip
monthly-full  user1  Sat Jan 01 00:00:03 PST 2011  domain1_2011_01_01_v00001.zip
monthly-full  user1  Tue Feb 01 00:00:01 PST 2011  domain1_2011_02_01_v00001.zip
Command list-backups executed successfully.
```

Note that this listing includes backups created automatically by a backup configuration. This feature is available only in Payara Server.

You can also view the full syntax and options of the subcommand by typing `asadmin help list-backups` at the command line.

Re-Creating the Domain Administration Server (DAS)

For mirroring purposes, and to provide a working copy of the DAS, you must have:

- One host (`olddashost`) that contains the original DAS.
- A second host (`apphost`) that contains a cluster with server instances running applications and catering to clients. The cluster is configured using the DAS on the first host.
- A third host (`newdashost`) where the DAS needs to be re-created in a situation where the first host crashes or is being taken out of service.

NOTE

You must maintain a backup of the DAS from the first host using the "`backup-domain`" subcommand as described in to-back-up-a-domain. You can automatically maintain a backup of the DAS using the automatic backups feature of Payara Server.

NOTE

Payara Server includes `asadmin` subcommands that simplify this procedure. If you are using Payara Server, see to-migrate-the-das.

To Migrate the DAS

The following steps are required to migrate the DAS from the first host (`olddashost`) to the third host (`newdashost`).

1. Install Payara Server on `newdashost` just as it was installed on `olddashost`.
   This is required so that the DAS can be properly restored on `newdashost` without causing path conflicts.
2. Use the `restore-domain` subcommand to restore the latest backup file onto `newdashost`.
   For example:

   ```
   asadmin> restore-domain --backupdir /net/backups.example.com/glassfish
   ```

   This example assumes that backups are stored in a network-accessible location. If this is not the case, manually copy the latest backup file from offline storage to a directory on `newdashost`.
   You can backup any domain. However, while re-creating the domain, the domain name should be same as the original.
3. Stop the domain on `olddashost`, if it is running.
4. Start the domain on `newdashost` by using the "`start-domain`" subcommand.
   For example:

   ```
   asadmin> start-domain domain1
   ```

5. If the domain on `olddashost` was centrally administered, set up centralized administration on `newdashost`.
   See enabling-centralized-administration-of-payara-server-instances in the Payara Server High Availability section for instructions.
6. Verify that instances on other hosts are visible to the new DAS on `newdashost`:

   ```
   asadmin> list-instances --long
   ```

7. Change the DAS host values for properties under the node on `apphost`.
   In the file `as-install/nodes/node-name/agent/config/das.properties` file, change the `agent.das.host` property value to refer to `newdashost` instead of `olddashost`.
8. Use the new DAS to restart clusters and standalone instances on `apphost`:

   Restarting the clustered and standalone instances on `apphost` triggers their recognition of the new DAS on `newdashost`.

   a. Use the `list-clusters` subcommand to list the clusters in the domain.
   b. Use the `stop-cluster` subcommand to stop each cluster.
   c. Use the `list-instances` subcommand to list the instances in the domain.
   d. Use the `restart-instance` subcommand to restart each standalone instance.
   e. Use the `start-cluster` subcommand to start each cluster.
      If the domain does not use centralized administration, use the `start-local-instance` subcommand to start the cluster instances on apphost.
9. Verify that instances on apphost are running:

```
asadmin> list-instances --long
```

10. Decommission and discontinue use of the DAS on olddashost.

Additional Domain Tasks

To Display Domain Uptime

Use the `uptime` subcommand in remote mode to display the length of time that the domain administration server (DAS) has been running since it was last started.

1. Ensure that the server is running.
   Remote subcommands require a running server.
2. Display uptime by using the "`uptime`" subcommand.

Example 3-17 Displaying the DAS Uptime

This example displays the length of time that the DAS has been running.

```
asadmin> uptime
Uptime: 1 Weeks, 4 days, 0 hours, 17 minutes, 14 seconds, Total milliseconds:
951434595
Command uptime executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help uptime` at the command line.

To Switch a Domain to Another Supported Java Version

Payara Server requires Java SE 11 as the underlying virtual machine for the Java platform (Java Virtual Machine or JVM machine).

NOTE	Do not downgrade to an earlier Java version after a domain has been created with a newer JVM machine. If you must downgrade your JVM machine, downgrade it only for individual domains.

1. If you have not already done so, download the desired Java SDK (not the JRE) and install it on your system.
2. Start the domain for which you are changing the JDK.
   Use the following format:

   ```
   as-install/bin/asadmin start-domain domain-name
   ```

   For a valid JVM installation, locations are checked in the following order:

   a. `domain.xml` (`java-home` inside `java-config`)
   b. `asenv.conf` (setting `AS_JAVA="path to java home"`)
      If a legal JDK is not found, a fatal error occurs and the problem is reported back to you.
   c. If necessary, change the JVM machine attributes for the domain.
      In particular, you might need to change the `JAVA_HOME` environment variable. For

example, to change the `JAVA_HOME` variable, type:

```
as-install/bin/asadmin set "server.java-config.java-home=path-to-java-home"
```

To Change the Administration Port of a Domain

Use the `set` subcommand in remote mode to change the administration port of a domain.

The HTTP port or the HTTPS port for administration of a domain is defined by the `--adminport` option of the "`create-domain`" subcommand when the domain is created. If this port must be reallocated for another purpose, change the port on which the DAS listens for administration requests.

1. Ensure that the server is running.
   Remote subcommands require a running server.
2. Set the port number to its new value.
   Use the "`set`" subcommand for this purpose.

   ```
   $ asadmin set
   server-config.network-config.network-listeners.network-listener.admin-
   listener.port=new-port-number
   ```

   The new-port-number is the new value that you are setting for the port number.

   NOTE After you set the port number to its new value, running the `list-domains` subcommand incorrectly reports that the DAS is not running. The `list-domains` subcommand reports the correct state again only after you stop and restart the domain as explained in the steps that follow.

3. Stop the domain, specifying the host on which the DAS is running and the old administration port number of the domain.
   You must specify the old port number because the DAS is still listening for administration requests on this port. If you omit the port number, the command fails

because the `stop-domain` subcommand attempts to contact the DAS through the new port number.

NOTE Only the options that are required to complete this task are provided in this step. For information about all the options for controlling the behavior of the domain, see the "`stop-domain`" help page.

```
$ asadmin --host host-name --port old-port-number stop-domain
```

- **host-name**: The name of the host on which the DAS is running. If you run the `stop-domain` subcommand on the host where the DAS is running, you must specify the actual host name and not `localhost`. If you specify `localhost`, the `stop-domain` subcommand fails.
- **old-port-number**: The value of administration port number of the domain before you changed it in the preceding step.

1. Start the domain.

NOTE Only the options that are required to complete this task are provided in this step. For information about all the options for controlling the behavior of the domain, see the "`start-domain`" help page.

```
$ start-domain [domain-name]
```

The domain-name is the name of the domain to start. If only one domain subdirectory is contained in the `domains` directory, you may omit this option.

Example 3-18 Changing the Administration Port of a Domain

This example changes the administration port of the domain `domain1` from 4848 to 4849. The DAS is running on the host `xk01.example.com`.

```
$ asadmin set
server-config.network-config.network-listeners.network-listener.admin-
listener.port=4849
server-config.network-config.network-listeners.network-listener.admin-
```

```
listener.port=4849
Command set executed successfully.
$ asadmin --host xk01.example.com --port 4848 stop-domain
Waiting for the domain to stop ....
Command stop-domain executed successfully.
$ asadmin start-domain
Waiting for domain1 to start .......................
Successfully started the domain : domain1
domain  Location: /export/payara7/glassfish/domains/domain1
Log File: /export/payara7/glassfish/domains/domain1/logs/server.log
Admin Port: 4849
Command start-domain executed successfully.
```

See Also

- "`create-domain`"
- "`set`"
- "`start-domain`"
- "`stop-domain`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help create-domain`
- `asadmin help set`
- `asadmin help start-domain`
- `asadmin help stop-domain`

Administering the Virtual Machine for the Java Platform

The Java Virtual Machine is an interpretive computing engine responsible for running the byte codes in a compiled Java program. The virtual machine translates the Java byte codes into the native instructions of the host machine. Payara Server, being a Java process, requires a virtual machine to run and support the Java applications running on it. JVM settings are part of an Payara Server configuration.

Administering JVM Options

The commands can be used to configure the JVM settings for Payara Server.

To Create JVM Options

Use the `create-jvm-options` subcommand in remote mode to create JVM options in the Java configuration element of the `domain.xml` file.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create JVM options by using the `create-jvm-options` subcommand. To create more than one JVM option, use a colon (:) to separate the options. If the JVM option itself contains a colon (:), use the backslash (\) to offset the colon delimiter.Information about properties for the subcommand is included in this help page.
3. To apply your changes, restart Payara Server.

Example 4-1 Creating JVM Options Example

This example sets multiple Java system properties.

```
asadmin> create-jvm-options -Dunixlocation=/root/example:
-Dvariable=\$HOME:
-Doption1=-value1
created 4 option(s)
Command create-jvm-options executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-jvm-options` at the command line.

To List JVM Options

Use the `list-jvm-options` subcommand in remote mode to list the existing JVM options.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List JVM options by using the `list-jvm-options` subcommand.

Example 4-2 Listing JVM Options

This example lists all JVM options.

```
asadmin> list-jvm-options
-Djava.security.auth.login.config=${com.sun.aas.instanceRoot}/config/login.conf
```

```
-XX: LogVMOutput
-XX: UnlockDiagnosticVMOptions
-Dcom.sun.enterprise.config.config_environment_factory_class=com.sun.enterprise.
config.serverbeans.AppserverConfigEnvironmentFactory
-Djavax.net.ssl.keyStore=${com.sun.aas.instanceRoot}/config/keystore.jks
-XX:NewRatio=2
-Djava.security.policy=${com.sun.aas.instanceRoot}/config/server.policy
-Djdbc.drivers=org.apache.derby.jdbc.ClientDriver
-Djavax.net.ssl.trustStore=${com.sun.aas.instanceRoot}/config/cacerts.jks
-client
-Djava.ext.dirs=${com.sun.aas.javaRoot}/lib/ext${path.separator}${com.sun.aas.ja
vaRoot}/jre/lib/ext${path.separator}${com.sun.aas.instanceRoot}/lib/ext${path.se
parator}${com.sun.aas.derbyRoot}/lib
-Xmx512m
-XX:LogFile=${com.sun.aas.instanceRoot}/logs/jvm.log
Command list-jvm-options executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-jvm-options` at the command line.

To Delete JVM Options

Use the `delete-jvm-options` subcommand in remote mode to delete JVM options from the Java configuration element of the `domain.xml` file.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List JVM options by using the `list-jvm-options` subcommand.
3. If necessary, notify users that the JVM option is being deleted.
4. Delete JVM options by using the `delete-jvm-options` subcommand. To remove more than one JVM option, use a colon (:) to separate the options. If the JVM option itself contains a colon, use the backslash (\) to offset the colon delimiter.
5. To apply your changes, restart Payara Server.

Example 4-3 Deleting a JVM Option

This example removes a single JVM option.

```
asadmin> delete-jvm-options -Doption1=-value1

deleted 1 option(s)
Command delete-jvm-options executed successfully.
```

Example 4-4 Deleting Multiple JVM Options

This example removes multiple JVM options.

```
asadmin> delete-jvm-options -Doption1=-value1:-Dvariable=\$HOME
deleted 2 option(s)
Command delete-jvm-options executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-jvm-options` at the command line.

Azul Payara Proprietary JVM Options

The following is a list of the proprietary JVM Options available for configuration:

Config File Encoding

Property: `fish.payara.configFileEncoding` **Aim**: Sets the domain.xml parser character encoding. The domain will fail to start if you are using characters not covered by UTF-8 in your configuration without this property configured. **Acceptable Values**: Any JDK supported file encoding charset

Configuring JDK Versions for JVM Options

It is possible to allow specific JVM options configured for a Payara Server instance to be applied only on specific JDK versions. The configuration is flexible since it allows the user to define minimum and maximum versions of all JDKs upon which each JVM option should be applied.

JDK versions must follow standard the standard JDK versioning scheme. The syntax that can be used allows for different levels of granularity in the specified JDK versions. Any of the following would be acceptable:

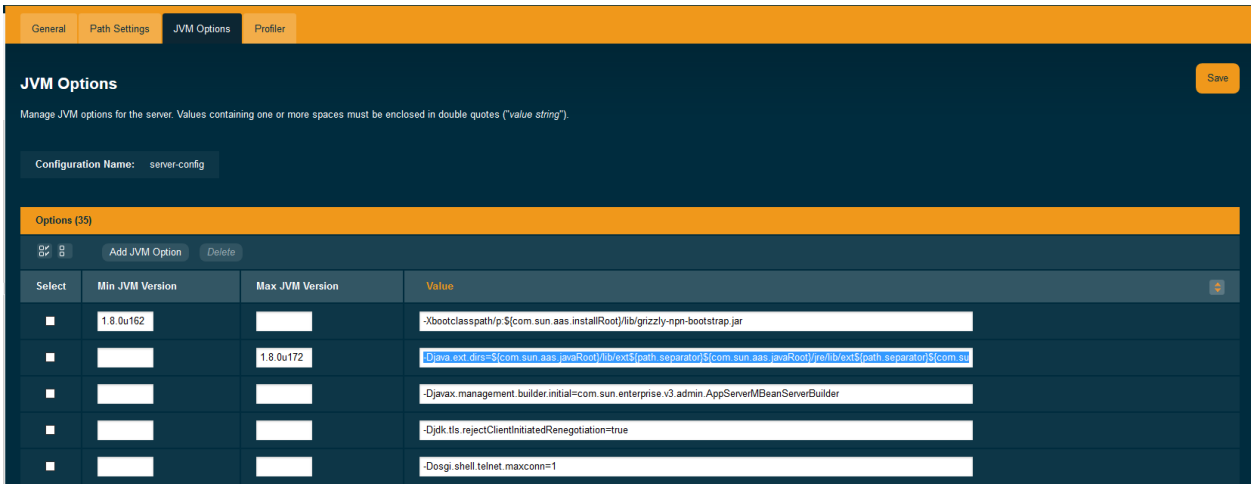
- `1.8.0.162`
- `1.8.0u162`
- `1.8.0`
- `1.8`

NOTE

If no minimum and/or maximum JDK versions are set for a particular JVM option, it shall be applied no matter which JDK is being used to run the server's instance.

Using the Admin Console

To configure the JDK versions for JVM options in the admin console, head to the **Configurations** ▯ **<instance configuration>** ▯ **JVM Settings** and on the **JVM Options** tab specify the versions in the table:



NOTE

You can omit either the minimum or maximum versions by leaving the fields blank.

Using Asadmin Commands

The `create-jvm-options` asadmin command allows the minimum and maximum JDK versions to be set in the following ways:

1. Use the `--min-jvm` and `--max-jvm` arguments to set the versions when running the command:

```
asadmin create-jvm-options --target=server-config --min-jvm=1.8.0u162 "-
Djava.awt.headless\=true"

asadmin create-jvm-options --target=server-config --max-jvm=1.8.0u172 "-XX
\:+UnlockDiagnosticVMOptions"
```

> **NOTE** When using these arguments to set multiple JVM options, the specified versions will be applied for each option.

2. Specify the versions using the `[MIN_VERSION | MAX_VERSION]` text syntax for the inclusive version range before each JVM option:

```
asadmin create-jvm-options --target=server-config "[1.8.0u162|]-
Djava.awt.headless\=true"

asadmin create-jvm-options --target=server-config "[|1.8.0u172]-XX
\:+UnlockDiagnosticVMOptions"

asadmin create-jvm-options --target=server-config "[1.8.0u168|1.8.0u172]-XX
\:NewRatio\=2"
```

The `delete-jvm-options` asadmin command does not require you to specify the JDK version when deleting a JVM option, though you can if you wish.

For example, given the option `[1.8.0.160|1.8.0.200]XX\:NewRatio=3`, either of the following commands are acceptable:

```
asadmin delete-jvm-options "XX\:NewRatio=3"

asadmin delete-jvm-options "[1.8.0.160|1.8.0.200]XX\:NewRatio=3"
```

To Generate a JVM Report

Use the `generate-jvm-report` subcommand in remote mode to generate a JVM report showing the threads (dump of a stack trace), classes, memory, and loggers for a specified instance, including the domain administration server (DAS). You can generate the following types of reports: summary (default), class, thread, log.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Generate the report by using the `generate-jvm-report` subcommand.

Example 4-5 Generating a JVM Report

This example displays summary information about the threads, classes, and memory.

```
asadmin> generate-jvm-report --type summary
```

```
Operating System Information:
Name of the Operating System: Windows XP
Binary Architecture name of the Operating System: x86, Version: 5.1
Number of processors available on the Operating System: 2
System load on the available processors for the last minute: NOT_AVAILABLE.
(Sum of running and queued runnable entities per minute).
.
,
.
user.home = C:\Documents and Settings\Jennifer
user.language = en
user.name = Jennifer
user.timezone = America/New_York
user.variant =
variable = \$HOME
web.home = C:\Preview\v3_Preview_release\distributions\web\target\
Payara\modules\web
Command generate-jvm-report executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help generate-jvm-report` at the command line.

Administration Console Features

This page documents features specific to the administration console, focusing on Payara Server enhancements.

Asadmin Recorder

This feature allows you to record the actions you take in the admin console as asadmin commands, aiding with automating or reproducing your setup across multiple Payara Server installations.

Starting and Stopping the Asadmin Recorder

You can enable or disable the asadmin recorder on its configuration page (detailed in the following section), or by using the following asadmin commands:

- enable-asadmin-recorder
- disable-asadmin-recorder
- set-asadmin-recorder-configuration

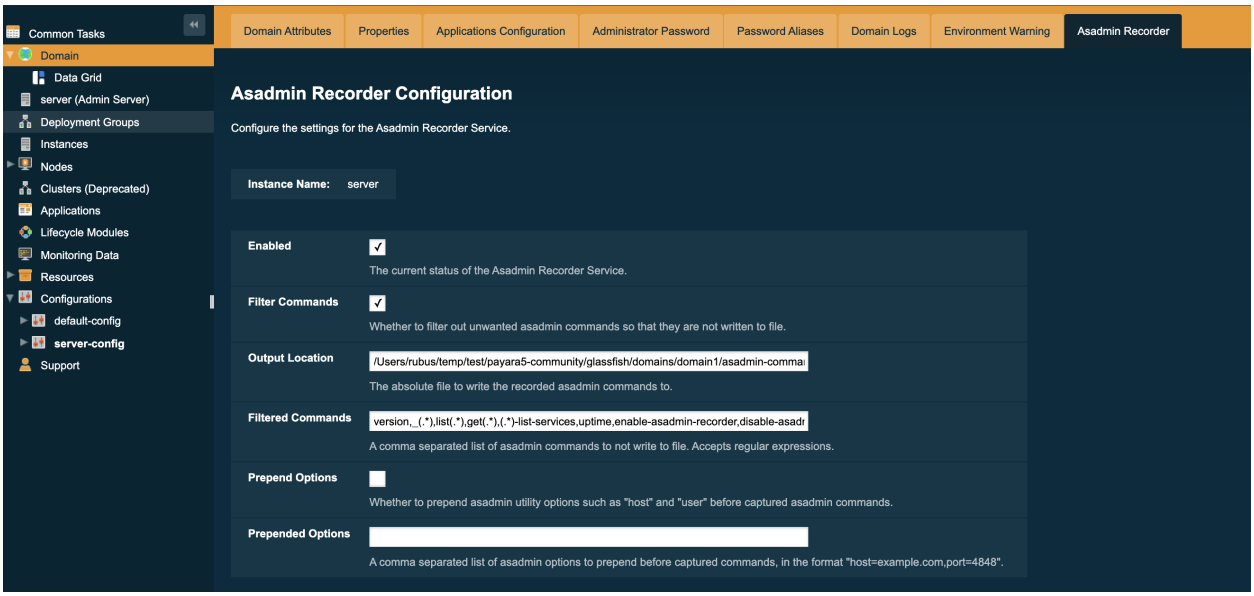
Once enabled or disabled, the asadmin recorder will remain enabled/disabled until specifically enabled or disabled again through the web console or using the appropriate

Asadmin CLI command.

This means that the asadmin recorder will remain enabled or disabled across server restarts.

Configuring the Asadmin Recorder

To configure the asadmin recorder, navigate to the following page in the admin console: *server* → *Asadmin Recorder*:



You will be presented with the following configuration options:

Enabled: Sets whether the asadmin recorder service should be enabled or disabled. ***Filter Commands***: Sets whether to exclude the `asadmin` commands and regular expressions listed in the *Filtered Commands* setting from being recorded. ***Output Location***: The absolute file path for the `asadmin` commands to be written to. This defaults to a file called *asadmin-commands.txt* in the domain directory. ***Filtered Commands***: A comma separated list of asadmin commands and regular expressions to be excluded from being recorded if the *Filter Commands* option is enabled.

The default regular expressions and asadmin commands set in the *Filtered Commands* option are a selection of commands and regular expressions that are typically not needed for any automation purposes, yet are still sometimes called by commands and

through navigation of the admin console.

If you remove any of these commands from the filtered commands list, or choose not to filter any commands at all, the asadmin commands script is liable to get filled up with these commands. Due to this behavior, it is advised that you only add to this list, only removing the defaults if you really need to.

Be sure to click on the *Save* button to have any changes you make take effect.

In addition to using the admin console, you can configure the asadmin recorder service using the `set-asadmin-recorder-configuration` command. See set-asadmin-recorder-configuration

Running the Generated Scripts

Documentation on running asadmin scripts can be found in to-run-a-set-of-asadmin-subcommands-from-a-file.

Asadmin Commands

The following is the list of available asadmin commands to configure the asadmin recorder:

- get-asadmin-recorder-configuration
- set-asadmin-recorder-configuration
- enable-asadmin-recorder
- disable-asadmin-recorder
- asadmin-recorder-enabled

Admin Console Auditing Service

This feature allows users to log all actions and operations executed via the admin console for auditing purposes.

Configuring the Auditing Service

All operations executed by users with access to the admin console will be translated to

events that correspond to the specific admin user that executed the operation and the `asadmin` command (and its parameters) that is equivalent to the operation being executed in the admin console. For example if the default `admin` user modifies the monitoring level of the *Web Services Container*, the following event will be generated:

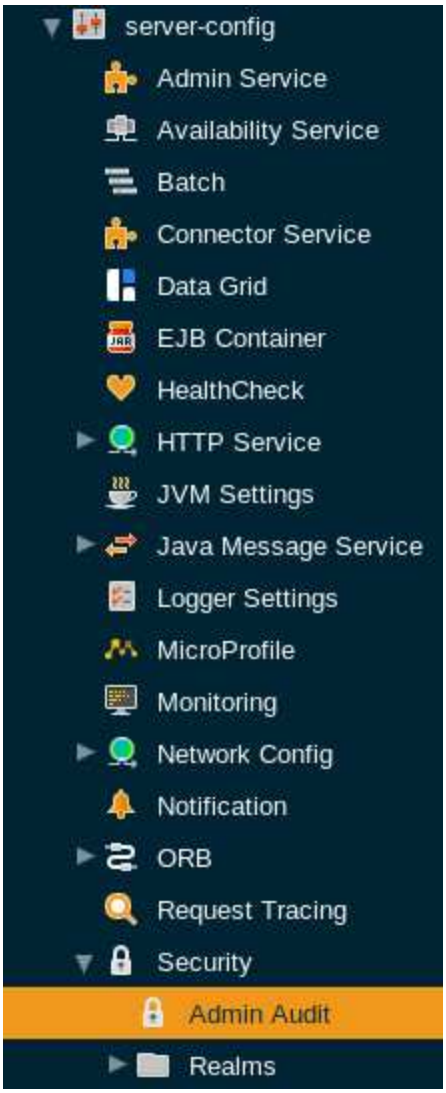
```
AUDIT - admin issued command set with parameters
{
    DEFAULT: [configs.config.server-config.monitoring-service.module-monitoring-
levels.web-services-container=HIGH]
}
```

The auditing service will relay these events to the "Notification Service". In this manner, administrators can be alerted of suspicious activity as soon as possible.

This service can be configured from the admin console and from the command line.

From the Admin Console

The auditing service configuration can be found in the web console under `Configurations` → `<configuration-name>` → `Security` → `Admin Audit`:



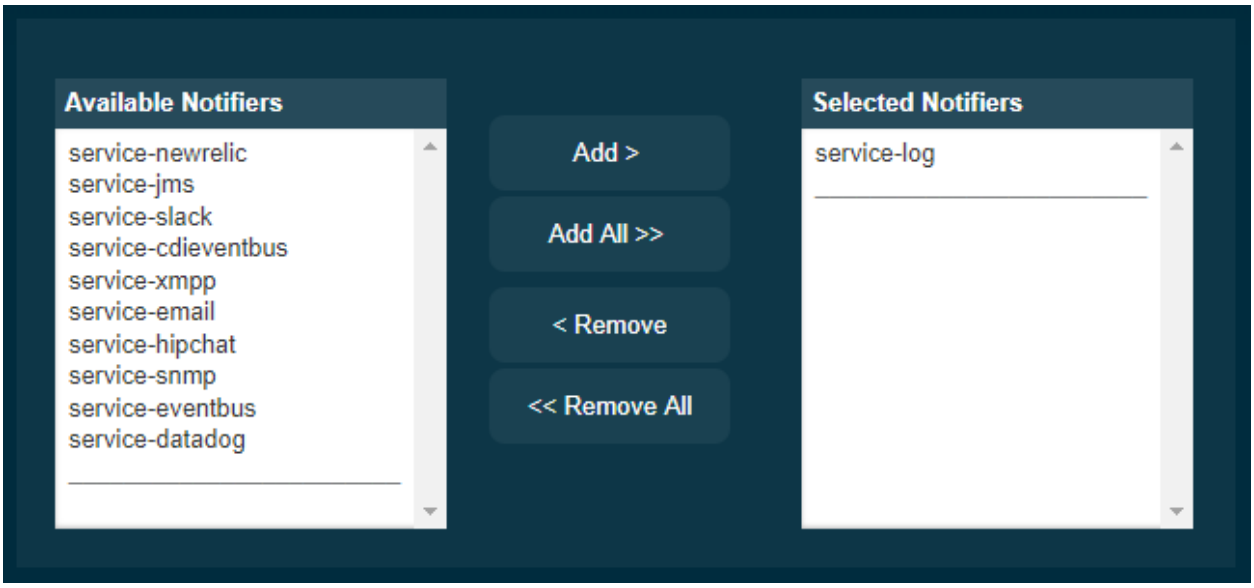
Once enabled, actions in the admin console that have a corresponding `asadmin` command will be sent to the "Notification service".

Check the **Enabled** option (and the **Dynamic** option too if you don't want to restart the domain) to switch the auditing service on.

The **Audit Level** options sets the level of operations that will be logged:

- **MODIFIERS**: (Default setting) All operations that modify the configuration of the server in some capacity.
- **ACCESSORS**: All operations query the configuration of the server in some capacity.
- **INTERNAL**: All existing operations, including internal operations of the server.

Aside from this configuration setting, you can also define which notifiers will be used to relay the admin console audit events by moving them to the **Selected Notifiers** box.



IMPORTANT	Keep in mind that for audit events to be relayed to the selected notifiers, both the Notification Service and each selected notifier must be enabled and configured beforehand.

TIP	You don't need to manually add each notifier on this screen. When enabling a notifier on its configuration screen, the server will **automatically** add it to the list of selected notifiers for the Admin audit service. This same result occurs when enabling the notifier using the appropriate *asadmin* command.

From the Command Line

The following is the list of available asadmin commands to configure the auditing service:

- set-admin-audit-configuration
- get-admin-audit-configuration

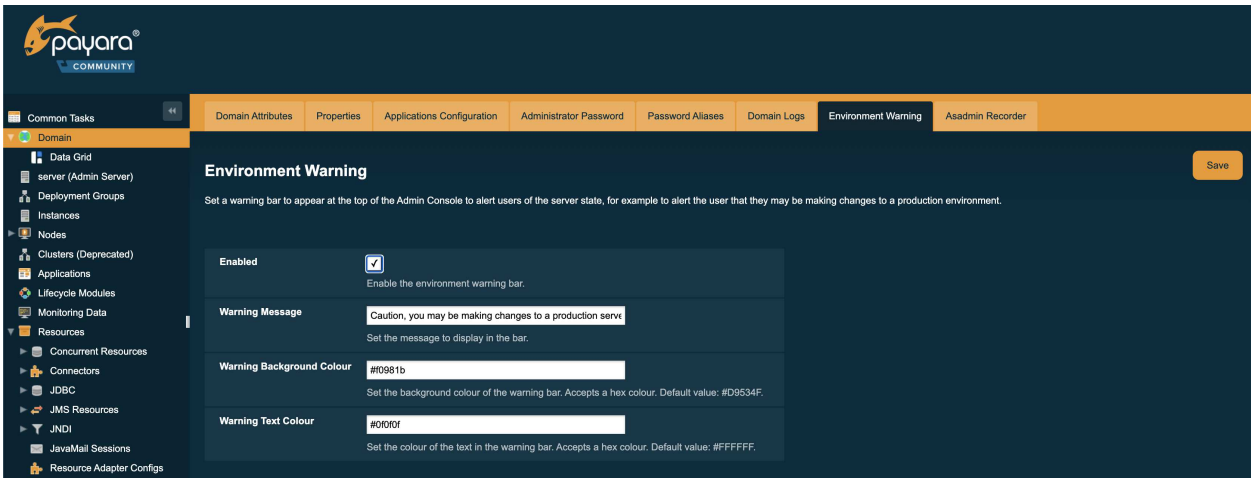
Admin Console Environment Warning

When working with multiple shared environments, it is common for restrictions to be in place on what modifications can be allowed for a given domain. For example, the test

environment cannot deviate significantly from the target production environment without potentially invalidating the test.

One way to ensure no accidental changes are made would be to revoke access to the admin console, but this has the downside of preventing engineers from seeing the current settings in the event that a bug is found.

As an alternative, Payara Server ships with a configurable warning bar to ensure any user will see a warning message in while viewing the console:



The warning can be configured in the admin console as seen in the screenshot above. There is a new "*Environment Warning*" tab in the "*Domain*" section which allows you to configure the text and colours of the warning message.

Configuring the Environment Warning via Asadmin

The following is the list of available asadmin commands to get and set the configuration the Environment Warning.

- set-environment-warning-configuration
- get-environment-warning-configuration

Administering Thread Pools

This chapter provides procedures for administering thread pools in the Payara Server environment by using the `asadmin` command-line utility.

Instructions for accomplishing these tasks by using the Administration Console are contained in the Administration Console online help.

About Thread Pools

The Virtual Machine for the Java platform ((Java Virtual Machine) or JVM machine) can support many threads of execution simultaneously. To help performance, Payara Server maintains one or more thread pools. It is possible to assign specific thread pools to connector modules, to network listeners, or to the Object Request Broker (ORB).

One thread pool can serve multiple connector modules and enterprise beans. Request threads handle user requests for application components. When Payara Server receives a request, it assigns the request to a free thread from the thread pool. The thread executes the client's requests and returns results. For example, if the request needs to use a system resource that is currently busy, the thread waits until that resource is free before allowing the request to use that resource.

Configuring Thread Pools

You can specify the minimum and maximum number of threads that are reserved for requests from applications. The thread pool is dynamically adjusted between these two values.

Default Thread Pool Size

The thread pool `thread-pool-1` starts with a minimum size of **2** instead of **1** (like in previous versions)

This includes the benefit of allowing users to execute tasks concurrently using the `ManagedExecutorService` and `ManagedScheduledExecutorService` components included with Java EE Concurrency Utilities (JSR-236). Without this change, new tasks submitted to these service components would just have been queued in a linear fashion.

NOTE	This minor optimization is configured on the default domain template (so all newly created domains inherit it).

To Create a Thread Pool

Use the `create-threadpool` subcommand in remote mode to create a thread pool.

The minimum thread pool size that is specified signals the server to allocate at least that many threads in reserve for application requests. That number is increased up to the maximum thread pool size that is specified. Increasing the number of threads available to a process allows the process to respond to more application requests simultaneously.

If one resource adapter or application occupies all the Payara Server threads, thread starvation might occur. You can avoid this by dividing the Payara Server threads into different thread pools.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create a new thread pool by using the "`create-threadpool`" subcommand. Information about options for the subcommand is included in this help page.
3. To apply your changes, restart Payara Server. See to-restart-a-domain.

NOTE	Restart is not necessary for thread pools used by the web container.

Example 5-1 Creating a Thread Pool

```
asadmin> create-threadpool --maxthreadpoolsize 100
--minthreadpoolsize 20 --idletimeout 2 threadpool-1
Command create-threadpool executed successfully
```

This example creates `threadpool-1`.

To List Thread Pools

Use the `list-threadpools` subcommand in remote mode to list the existing thread pools.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the existing thread pools by using the "`list-threadpools`" subcommand.

Example 5-2 Listing Thread Pools

```
asadmin> list-threadpools
threadpool-1
Command list-threadpools executed successfully
```

This example lists the existing thread pools.

To Update a Thread Pool

Use the `set` subcommand to update the values for a specified thread pool.

1. List the existing thread pools by using the "`list-threadpools`" subcommand.
2. Modify the values for a thread pool by using the "`set`" subcommand.
   The thread pool is identified by its dotted name.
3. To apply your changes, restart Payara Server. See to-restart-a-domain.

NOTE	Restart is not necessary for thread pools used by the web container.

Example 5-3 Updating a Thread Pool

```
asadmin> set server.thread-pools.thread-pool.http-thread-pool.max-thread-pool-size
=8
Command set executed successfully
```

This example sets the `max-thread-pool-size` from its previous value to 8.

To Delete a Thread Pool

Use the `delete-threadpool` subcommand in remote mode to delete an existing thread pool. Deleting a thread pool will fail if that pool is referenced by a network listener.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the existing thread pools by using the "`list-threadpools`" subcommand.
3. Delete the specified thread pool by using the "`delete-threadpool`" subcommand.
4. To apply your changes, restart Payara Server. See to-restart-a-domain.

NOTE | Restart is not necessary for thread pools used by the web container.

Example 5-4 Deleting a Thread Pool

```
asadmin> delete-threadpool threadpool-1
Command delete-threadpool executed successfully
```

This example deletes `threadpool-1`.

EJB Thread Pool

The EJB Thread Pool is for prepared EJB beans which are ready for usage, this means there is no instantiation and initialization overhead when the bean is used. It can also be used to limit the number of concurrent beans in use.

An individual bean can specify a configuration that overrides the configuration of the EJB container in the `<bean-pool>` element of the `glassfish-ejb-jar.xml` file.

EJB Thread Pool Configuration Properties

The EJB Thread Pool has the following properties configured by default.

Setting Name	Default Value	Description
Core Pool Size	16	The minimum number of threads in the EJB container thread pool
Max Pool Size	32	The maximum number of threads in the EJB container thread pool
Keep Alive Seconds	60	The time, in seconds, which threads in excess of thread-core-pool-size are deleted
Queue Capacity	Integer.MAX_VALUE (2,147,483,647)	The size, in bytes, of the thread pool queue which stores new requests while new threads are created if more than thread-core-pool-size number of threads are running.
Allow Core Thread Timeout	false	All threads, including core threads, are subject to termination after thread-keep-alive-seconds

Setting Name	Default Value	Description
Prestart All Core Threads	false	Threads in the EJB container thread pool are started even if no requests have arrived, and the pool has a non-empty queue

Configuring the EJB Thread Pool

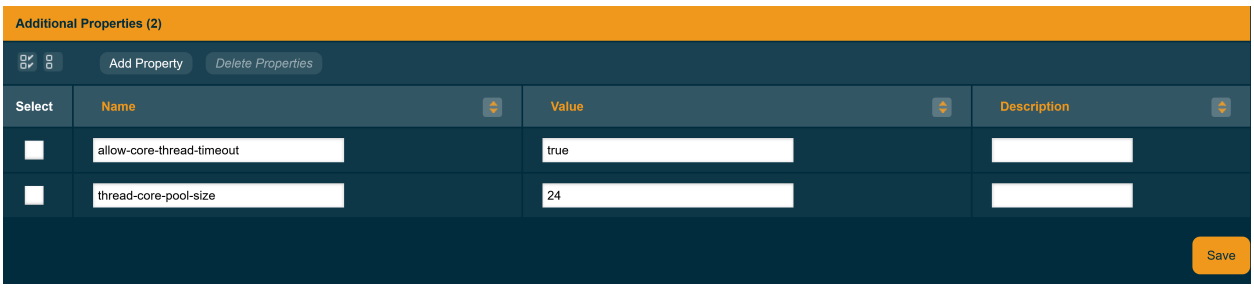
The EJB thread pool can be configured with either the admin console or the asadmin CLI, by making use of the following configuration properties.

Property Name	Value Constraints
thread-core-pool-size	Cannot be less than 0, larger than max int, and must be larger than `thread-max-pool-size`
thread-max-pool-size	Cannot be less than 1 or greater than max int
thread-keep-alive-seconds	Cannot be less than 0 or greater than max int
thread-queue-capacity	Cannot be less than 1 or greater than max int
allow-core-thread-timeout	Must be true or false
prestart-all-core-threads	Must be true or false

Using the Admin Console

To configure the EJB thread pool using the admin console, go to Configuration [instance-configuration] EJB Container and scroll down to see `Additional Properties`.

Add a new additional property, set the Name to the property name from the table above of the property you want to set, and set the Value to a valid value for that property.



Using the Asadmin CLI

To configure the EJB thread pool with the asadmin CLI, execute the following set command, replacing the configName with the name of your configuration, the property name with a property name from the table above and a valid value for that property.

```
asadmin set configs.config.${configName}.ejb-container.property.${propertyName}=
${propertyValue}
```

Administering the Logging Service

This chapter provides instructions on how to configure logging and how to view log information in the Payara Server environment.

Instructions for accomplishing these tasks and editing logging service properties in the Administration Console are available from the Administration Console online help.

About Logging

Logging is the process by which Payara Server captures information about events that occur, such as configuration errors, security failures, or server malfunction. This data is recorded in log files and is usually the first source of information when problems occur. Analyzing the log files can help you to follow events that occur in the server runtime and determine the overall health of the server.

Although application components can use the Apache Commons Logging Library to record messages, the platform standard JSR 47 Logging API is recommended for better log configuration.

For more information about Logging in Java, see:

- JSR 47 Specification
- Logging Overview

Log Files

Payara Server log records are captured in the server log file, which captures information about the operation of a server instance running in the domain. Each instance, managed server instance (that is, each cluster member), and the domain administration server

(DAS) has an individual server log file.

Log File Names and Locations

In a Payara Server domain, log files have the following name and location by default:

Instance	Default Log File Name and Location
DAS	`domain-dir/logs/server.log`
Server instance	`instance-dir/logs/server.log`
Cluster instance	`instance-dir/logs/server.log`

For example, in a domain hosted on a given machine that includes a cluster with two managed servers (`ClusterServer1` and `ClusterServer1`) and a standalone instance (`StandaloneServer`), the log files might be arranged in the following directory structure. In this directory structure, the `server.log` file for the DAS is located in `domain-dir/logs`.

```
as-install-parent directory
     glassfish/
         domains/
             domain-dir/
                         logs/
                            server.log
         nodes/
             hostname/
                         ClusterServer1/
                                     logs/
                                        server.log
                         ClusterServer2/
                                     logs/
                                        server.log
                         StandaloneServer/
                                     logs/
                                        server.log
```

You can change the default name or location of a log file by modifying the logging properties file for the corresponding instance, described in to-change-the-name-and-location-of-the-log-file.

Log Records

Payara Server writes log records in either the Oracle Diagnostics Logging (ODL) format

or the Uniform Log Formatter (ULF) format. The default format is ODL. For information about changing the format, see setting-the-log-file-format.

A custom formatter can also be used by specifying the fully qualified name of a class that extends the `java.util.logging.Formatter` class. The custom formatter class can be packaged in a JAR file and placed in the `domain-dir/lib/ext` directory, or implemented as a Hundred-Kilobyte Kernel (HK2) service and the JAR containing the formatter implementation placed in the `as-install/modules` directory.

Log records formatted in the ODL format are displayed as follows:

```
[yyyy-mm-ddThh:mm:ss.SSS-Z][ProductName-Version][Log Level][Message
ID][LoggerName][Key Value Pairs][[Message]]
```

- `[` and `]` mark the beginning and end of fields in the record. Some fields may be empty.
- `yyyy-mm-dd`T`hh:mm:ss.SSSS-Z` represents the date and time that the record was created. For example: `2013-04-12T08:08:30.154-0700`
- ProductName-Version represents the current version of Payara Server. For example: `payara`
- Log Level represents the log level.

  You can set any of the following values: `EMERGENCY`, `ALERT`, `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, and `FINEST`.

  The default is `INFO`.
- Message ID consists of a module code that indicates the origin of the message and the message number. For example: `AS-WEB-GLUE-00172`
- LoggerName represents a hierarchical logger namespace that identifies the source of the log module. For example: `jakarta.enterprise.web`
- Key Value Pairs represents pairs of key names and values. For example: `timeMillis: 1365779310154`

- Message represents the text of the log message.

  For multiline messages, `\[[` marks the end of the first line, and `]]` marks the end of the message. Multiline messages start on a new line after the message header and are indented.

The following is an example of a log record formatted in the ODL format:

```
[2013-04-12T08:08:30.154-0700] [Payara Server 7.2.0] [INFO] [AS-WEB-GLUE-00172]
[jakarta.enterprise.web] [tid: _ThreadID=217 _ThreadName=admin-listener(21)]
[timeMillis: 1365779310154] [levelValue: 800] [[
  Loading application [payroll] at [/payroll]]]
```

Log records formatted in the ULF format are displayed as follows:

```
[#|yyyy-mm-ddThh:mm:ss.SSS-Z|Log Level|ProductName-Version|LoggerName|Key Value
Pairs|Message|#]
```

- `[ and]` mark the beginning and end of the record.
- The vertical bar (`|`) separates the fields of the record.
- yyyy-mm-dd`T`hh:mm:ss.SSSS-Z represents the date and time that the record was created. For example: `2013-04-18T09:27:44.315-0700`
- Log Level represents the log level. You can set any of the following values: `EMERGENCY`, `ALERT`, `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, and `FINEST`. The default is `INFO`.
- ProductName-Version represents the current version of Payara Server. For example: `Payara Server 7.2.0`
- LoggerName represents a hierarchical logger namespace that identifies the source of the log module. For example: `jakarta.enterprise.web.core`
- Key Value Pairs represents pairs of key names and values and can include a message ID. For example: `_MessageID=AS-WEB-CORE-00306`.

  Message IDs are included for all Payara Server `EMERGENCY`, `ALERT`, `SEVERE`, and `WARNING` messages, and for many `INFO` messages.

- Message represents the text of the log message. Multiline messages start on a new line after the message header and are indented.

The following is an example of a log record formatted in the ULF format:

```
[#|2013-04-18T09:27:44.315-0700|INFO|Payara Server
7.2.0|jakarta.enterprise.web.core|
_ThreadID=15;_ThreadName=RunLevelControllerThread-
1366302462682;_TimeMillis=1366302464315;_LevelValue=800;_MessageID=AS-WEB-CORE-
00306;| Setting JAAS app name glassfish-web|#]
```

The Administration Console presents log records in a more readable display. For information about viewing log records using the Administration Console, see viewing-log-records.

Log Rotation

By default, when a log file grows to 2 MB, Payara Server renames (rotates) the file to incorporate a timestamp and creates a new log file. The log file is renamed as `server.log` date, where date is the date and time that the file was rotated.

You can configure the logging service to change the default settings for log file rotation, as explained in setting-log-file-rotation.

Logger Namespaces

Payara Server provides a logger for each of its modules. The following list is an example of the logger namespaces in a server instance as they appear when using the `list-log-levels` subcommand.

```
ShoalLogger     <CONFIG>
com.hazelcast   <WARNING>
com.sun.enterprise.server.logging.GFFileHandler <ALL>
com.sun.enterprise.server.logging.SyslogHandler <ALL>
jakarta.mail    <INFO>
java.util.logging.ConsoleHandler        <FINEST>
javax.enterprise.resource.corba <INFO>
javax.enterprise.resource.javamail      <INFO>
javax.enterprise.resource.jdo   <INFO>
javax.enterprise.resource.jms   <INFO>
javax.enterprise.resource.jta   <INFO>
javax.enterprise.resource.resourceadapter       <INFO>
javax.enterprise.resource.sqltrace      <FINE>
javax.enterprise.resource.webcontainer.jsf.application  <INFO>
```

```
javax.enterprise.resource.webcontainer.jsf.config       <INFO>
javax.enterprise.resource.webcontainer.jsf.context      <INFO>
javax.enterprise.resource.webcontainer.jsf.facelets     <INFO>
javax.enterprise.resource.webcontainer.jsf.lifecycle    <INFO>
javax.enterprise.resource.webcontainer.jsf.managedbean  <INFO>
javax.enterprise.resource.webcontainer.jsf.renderkit    <INFO>
javax.enterprise.resource.webcontainer.jsf.resource     <INFO>
javax.enterprise.resource.webcontainer.jsf.taglib       <INFO>
javax.enterprise.resource.webcontainer.jsf.timing       <INFO>
javax.enterprise.system.container.ejb   <INFO>
javax.enterprise.system.container.ejb.mdb       <INFO>
javax.enterprise.system.container.web   <INFO>
javax.enterprise.system.core.classloading       <INFO>
javax.enterprise.system.core.config     <INFO>
javax.enterprise.system.core    <INFO>
javax.enterprise.system.core.security   <INFO>
javax.enterprise.system.core.selfmanagement     <INFO>
javax.enterprise.system.core.transaction        <INFO>
javax.enterprise.system <INFO>
javax.enterprise.system.ssl.security    <INFO>
javax.enterprise.system.tools.admin     <INFO>
javax.enterprise.system.tools.backup    <INFO>
javax.enterprise.system.tools.deployment.common <WARNING>
javax.enterprise.system.tools.deployment.dol    <WARNING>
javax.enterprise.system.tools.deployment        <INFO>
javax.enterprise.system.util    <INFO>
javax.enterprise.system.webservices.registry    <INFO>
javax.enterprise.system.webservices.rpc <INFO>
javax.enterprise.system.webservices.saaj        <INFO>
javax   <INFO>
javax.mail      <INFO>
javax.org.glassfish.persistence <INFO>
org.apache.catalina     <INFO>
org.apache.coyote       <INFO>
org.eclipse.persistence.session <INFO>
org.glassfish.admingui  <INFO>
org.glassfish.naming    <INFO>
org.glassfish.wasp      <INFO>
org.jvnet.hk2.osgiadapter       <INFO>
```

For information about how to display logger namespaces and log levels, see to-list-log-levels. For information about how to display a list of loggers and logger details, see to-list-loggers.

Logging Targets

Each instance in a Payara Server domain has a dedicated log file, and each instance and cluster has its own logging properties file. To configure logging for an instance or a cluster, Payara Server allows you target specific log files or logging properties files when you do the following:

- Set global or module-specific log levels
- Rotate log files or compress them into a ZIP archive
- Change logging property attributes
- List log levels or log attributes

The following subcommands optionally accept a target specification. A target can be a configuration name, server name, cluster name, or instance name, and is specified as either an operand or as a value passed using the `--target` option. If no target is specified when using any of these subcommands, the default target is the DAS.

Subcommand	Description	Target Specification
"`collect-log-files`"	Collects all available log files into a ZIP archive.	`--target=target-name`
"`list-log-attributes`"	Lists logging attributes in the logging properties file.	target-name operand
"`list-log-levels`"	Lists the loggers in the logging properties file and their log levels	target-name operand
"`rotate-log`"	Rotates the log file by renaming it and creating a new log file to store new messages.	`--target=target-name`
"`set-log-attributes`"	Sets the specified logging attributes in the logging properties file.	`--target=target-name`
"`set-log-file-format`"	Sets the log file formatter.	`--target=target-name`
"`set-log-levels`"	Sets the log level for one or more loggers listed in the logging properties file.	`--target=target-name`

Logging Properties

The DAS as well as each configuration, instance, and cluster has its own set of logging properties that are maintained in individual configuration files. A logging properties file is named `logging.properies` and includes the following information:

- Log file name and location
- Logger names and levels
- Properties for custom handlers
- Log rotation and logger format properties

By default, in a Payara Server domain, logging properties files are created in the following locations:

Target	Default Location of Logging Properties File
DAS	`domain-dir/config/logging.properties`
A configuration	`domain-dir/config/<config-name>/logging.properties`, where `<config-name>` represents the name of a configuration that is shared by one or more instances or clusters.
An instance	`domain-dir/config/<instance-name>-config/logging.properties`, where `<instance-name>` represents the name of the instance.
A cluster	`domain-dir/config/<cluster-name>-config/logging.properties`, where `<cluster-name>` represents the name of the cluster.

Changing the Name and Location of Logging Service Files

This section explains how to change the name and location of the following logging service files:

- Log file
- Logging properties file

To Change the Name and Location of the Log File

To change the name and location of the log file, first use the `list-log-attributes` subcommand to obtain the current log attribute setting for the log file name and location.

Then use the `set-log-attributes` subcommand to specify the new name or

location. The default target for these two subcommands is the DAS. However, you can optionally specify one of the following targets:

- Configuration name — to target all instances or clusters that share a specific configuration name.
- Server name — to target only a specific server.
- Instance name — to target only a specific instance.
- Cluster name — to target only a specific cluster.

1. Ensure that the target server or cluster is running. Remote subcommands require a running server.
2. Use the "`list-log-attributes`" subcommand in remote mode to obtain the current log attribute settings.

   The name and location of the log file is set with the `com.sun.enterprise.server.logging.GFFileHandler.file` attribute of the logging properties file. Optionally you can target a configuration, server, instance, or cluster.

   If you do not specify a target, the log attribute settings for the DAS are displayed.
3. Use the "`set-log-attributes`" subcommand in remote mode to define a custom name or location of the log file.

   If you do not specify a target, the log file for the DAS is targeted by default. If you target a cluster, the name of the cluster log file for each member instance can be changed (the server log file name cannot).

Example - to Change the Name and Location of the Log File

This example changes the name of the cluster log file for `Cluster1` to `cluster1.log`. `Cluster1` has two server instances: `ClusterServer1` and `ClusterServer2`.

```
asadmin list-log-attributes Cluster1
```

```
com.sun.enterprise.server.logging.GFFileHandler.alarms <false>
com.sun.enterprise.server.logging.GFFileHandler.file
<${com.sun.aas.instanceRoot}/logs/server.log>
com.sun.enterprise.server.logging.GFFileHandler.flushFrequency  ¬
.
.
.
log4j.logger.org.hibernate.validator.util.Version <warn>
Command list-log-attributes executed successfully.
asadmin> set-log-attributes --target Cluster1
com.sun.enterprise.server.logging.GFFileHandler.file=
${com.sun.aas.instanceRoot}/logs/cluster1.log
com.sun.enterprise.server.logging.GFFileHandler.file
logging attribute set with value ${com.sun.aas.instanceRoot}/logs/cluster1.log
These logging attributes are set for Cluster1.
ClusterServer1 :
com.sun.enterprise.server.logging.GFFileHandler.file
logging attribute set with value ${com.sun.aas.instanceRoot}/logs/cluster1.log
These logging attributes are set for Cluster1.

ClusterServer2 :
com.sun.enterprise.server.logging.GFFileHandler.file
logging attribute set with value ${com.sun.aas.instanceRoot}/logs/cluster1.log
These logging attributes are set for Cluster1.

Command set-log-attributes executed successfully.
```

You can view the full syntax and options of these subcommands by typing `asadmin help list-log-levels` and `asadmin help set-log-attributes` at the command line.

To Change the Name and Location of the Logging Properties File

You can set the name and location of the logging properties file by setting the `java.util.logging.config.file` system property. By setting this system property, you can have a single logging properties file that is used by all instances running on the same host.

NOTE

Setting the `java.util.logging.config.file` system property causes all other Payara Server logging properties files on the host to be overridden.

1. Set the `java.util.logging.config.file` system property.

   For example, you can use the following command:

```
java -Djava.util.logging.config.file=properties_file
```

Alternatively, you can use the Administration Console to set this system property.

2. To apply your change, restart Payara Server.

Example - to Change the Name and Location of the Logging Property File

The following example changes the location of the logging properties file to `/space/mylogging/logging.properties`:

```
java -Djava.util.logging.config.file=/space/mylogging/logging.properties
```

Setting Log Levels

The log level determines the granularity of the message that is logged, from error only (`EMERGENCY`) to detailed debug (`FINEST`). The following values apply: `EMERGENCY`, `ALERT`, `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, and `FINEST`.

These log levels are hierarchically inclusive, which means that if you set a particular log level, such as INFO, the messages that have log levels above that level (`EMERGENCY`, `ALERT`, `SEVERE`, and `WARNING`,) are also included. If you set the log level to the lowest level, `FINEST`, your output includes all the messages in the file. The default setting is `INFO`.

You can list current log levels of all loggers specified in the `logging properties` file. In some cases, loggers that have not been created by the respective containers will also appear in the list.

You can set log levels that are global or logger-specific. When you set a global log level, the log level goes into effect for all loggers. If you set the log level for a specific logger that is different from the global log level, the logger-specific setting takes precedence. In addition, when setting log levels, you can target a configuration, server, instance, or cluster.

Because setting log levels is a dynamic operation, you do not need to restart Payara

Server for changes to take effect.

Setting either global or logger-specific log levels is done by using the `set-log-levels` subcommand. Listing log levels is done by using the `list-log-levels` subcommand.

To List Log Levels

Payara Server provides the means to list all loggers and their log levels. Listing the loggers provides a convenient means to view current loggers and log levels either prior to or after making log level changes.

Use the `list-log-levels` subcommand in remote mode to list the modules and their current log levels. The default target for this subcommand is the DAS. However, you can optionally specify one of the following targets:

- Configuration name — to target all instances or clusters that share a specific configuration name.
- Server name — to target a specific server.
- Instance name — to target a specific instance.
- Cluster name — to target a specific cluster.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. List the existing module loggers and log levels by using the "`list-log-levels`" subcommand.

Example - to List Log Levels Part 1

This example shows a partial list of the existing loggers and their log levels in the DAS.

```
asadmin list-log-levels

jakarta.enterprise.system.tools.admin <INFO>
java.util.logging.ConsoleHandler <FINEST>
jakarta.enterprise.system.container.web <INFO>
jakarta.enterprise.system.util <INFO>
jakarta.enterprise.resource.webcontainer.jsf.timing <INFO>
javax <INFO>
jakarta.enterprise.resource.corba <INFO>
...
Command list-log-levels executed successfully.
```

Example - to List Log Levels Part 2

This example shows a partial list of the loggers and log levels for the instance `MyServer2`.

```
asadmin list-log-levels MyServer2

java.util.logging.ConsoleHandler <FINEST>
jakarta.enterprise.resource.corba  <INFO>
jakarta.enterprise.resource.javamail   <INFO>
jakarta.enterprise.resource.jdo <INFO>
jakarta.enterprise.resource.jms <INFO>
jakarta.enterprise.resource.jta <INFO>
jakarta.enterprise.resource.resourceadapter <INFO>
jakarta.enterprise.resource.sqltrace <FINE>
...
Command list-log-levels executed successfully.
```

You can view the full syntax and options of the subcommand by typing `asadmin help list-log-levels` at the command line.

To Set the Global Log Level

The global log level specifies the events that are logged across all loggers. The default level for messages output to the console is `INFO` (which also includes `EMERGENCY`, `ALERT`, `SEVERE`, and `WARNING` messages).

Use the `set-log-levels` subcommand in remote mode to set the global log level. The default target for this subcommand is the DAS. However, you can optionally specify one of the following targets using the `--target` option:

- Configuration name — to target all instances or clusters that share a specific configuration name.
- Server name — to target a specific server.
- Instance name — to target a specific instance.
- Cluster name — to target a specific cluster.

1. Ensure that the target server or cluster is running.
2. Set the global log level by using the "`set-log-levels`" subcommand, specifying the log level of the `java.util.logging.ConsoleHandler` logger.

The `ConsoleHandler` has a separate log level setting that limits the messages that are displayed. For example:

```
java.util.logging.ConsoleHandler <FINEST>
```

Example - to Set the Global Log Level

By setting the log level of the `ConsoleHandler`, you set the global log level for all loggers. This example sets the global log level in the DAS to `INFO`:

```
asadmin set-log-levels java.util.logging.ConsoleHandler=INFO

java.util.logging.ConsoleHandler package set with log level INFO.
These logging levels are set for server.

Command set-log-levels executed successfully.
```

You can view the full syntax and options of the subcommand by typing `asadmin help set-log-levels` at the command line.

To Set Module Log Levels

A module log level specifies the events that are logged for a particular logger. The default level for messages output to the console is `INFO` (which also includes `EMERGENCY`, `ALERT`, `SEVERE`, and `WARNING` messages). The global log level is overridden by a module-specific log level.

By default, the module log level is set to `FINE`. The lines for the loggers in the logging properties file might look like this:

```
jakarta.enterprise.system.tools.level=FINE
jakarta.enterprise.system.container.ejb.level=FINE
jakarta.enterprise.system.core.security.level=FINE
jakarta.enterprise.system.tools.admin.level=FINE
jakarta.enterprise.level=FINE
jakarta.enterprise.system.container.web.level=FINE
```

Because setting log levels is a dynamic operation, you do not need to restart Payara Server for changes to take effect.

1. Ensure that the target server or cluster is running.

   Remote subcommands require a running server.

2. List the existing module loggers and their log levels by using the "`list-log-levels`" subcommand.
3. Set the log level for a module by using the "`set-log-levels`" subcommand.

   Your choices are `EMERGENCY`, `ALERT`, `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, and `FINEST`.

Example - to Set the Module Log Levels Part 1

This example sets the log level for the web container logger to WARNING on the target instance `ManagedServer1`:

```
asadmin set-log-levels --target ManagedServer1

jakarta.enterprise.system.container.web=WARNING
jakarta.enterprise.system.container.web package set with log level WARNING.
These logging levels are set for ManagedServer1.
ManagedServer1 :
jakarta.enterprise.system.container.web package set with log level WARNING.
These logging levels are set for ManagedServer1.

Command set-log-levels executed successfully.
```

Example - to Set the Module Log Levels Part 2

The following example sets the log level for security and web container loggers in the DAS.

```
asadmin set-log-levels jakarta.enterprise.system.core.security=FINE:

jakarta.enterprise.system.container.web=WARNING
jakarta.enterprise.system.container.web package set with log level WARNING.
jakarta.enterprise.system.core.security package set with log level FINE.
These logging levels are set for server.

Command set-log-levels executed successfully.
```

You can view the full syntax and options of the subcommand by typing `asadmin help set-log-levels` at the command line.

Setting the Log File Format

You can set the format for log records in log files.

To Set the Log File Format

Use the `set-log-file-format` subcommand in remote mode to set the formatter used by Payara Server to format log records in log files. You can also use the `set-log-attributes` subcommand.

Log formats for all server instances in a cluster will be the same. For information about log formats, see log-records.

NOTE	Changing the log format forces log rotation to avoid mixed format in the same file.

1. Ensure that the target server or cluster is running. Remote commands require a running server.
2. Set the formatter by using the "`set-log-file-format`" subcommand.
3. To apply your change, restart Payara Server.

Example - to Set the Log File Format Part 1

This example sets the log file format to `ULF` for standalone instance `ManagedServer1` using the `set-log-file-format` subcommand.

```
asadmin set-log-file-format --target ManagedServer1 ulf

The log file formatter is set to
com.sun.enterprise.server.logging.UniformLogFormatter for instance ManagedServer1.
Command set-log-file-format executed successfully.
```

Example - to Set the Log File Format Part 2

This example sets the log file format to `ULF` for standalone instance `ManagedServer1` using the `set-log-attributes` subcommand.

```
asadmin set-log-attributes --target ManagedServer1
com.sun.enterprise.server.logging.GFFileHandler.formatter=ulf

com.sun.enterprise.server.logging.GFFileHandler.formatter logging attribute value
```

```
set to ulf.
The logging attributes are saved successfully for ManagedServer1-config.
Command set-log-attributes executed successfully.
```

You can view the full syntax and options of the `set-log-file-format` subcommand by typing `asadmin help set-log-file-format` at the command line. You can view the full syntax and options of the `set-log-attributes` subcommand by typing `asadmin help set-log-attributes` at the command line.

ANSI Coloured Logging

Payara Server supports the use of ANSI coloured log entries when running in verbose mode under the uniform log format (ULF).

To enable ANSI colours run the following command using the Asadmin CLI:

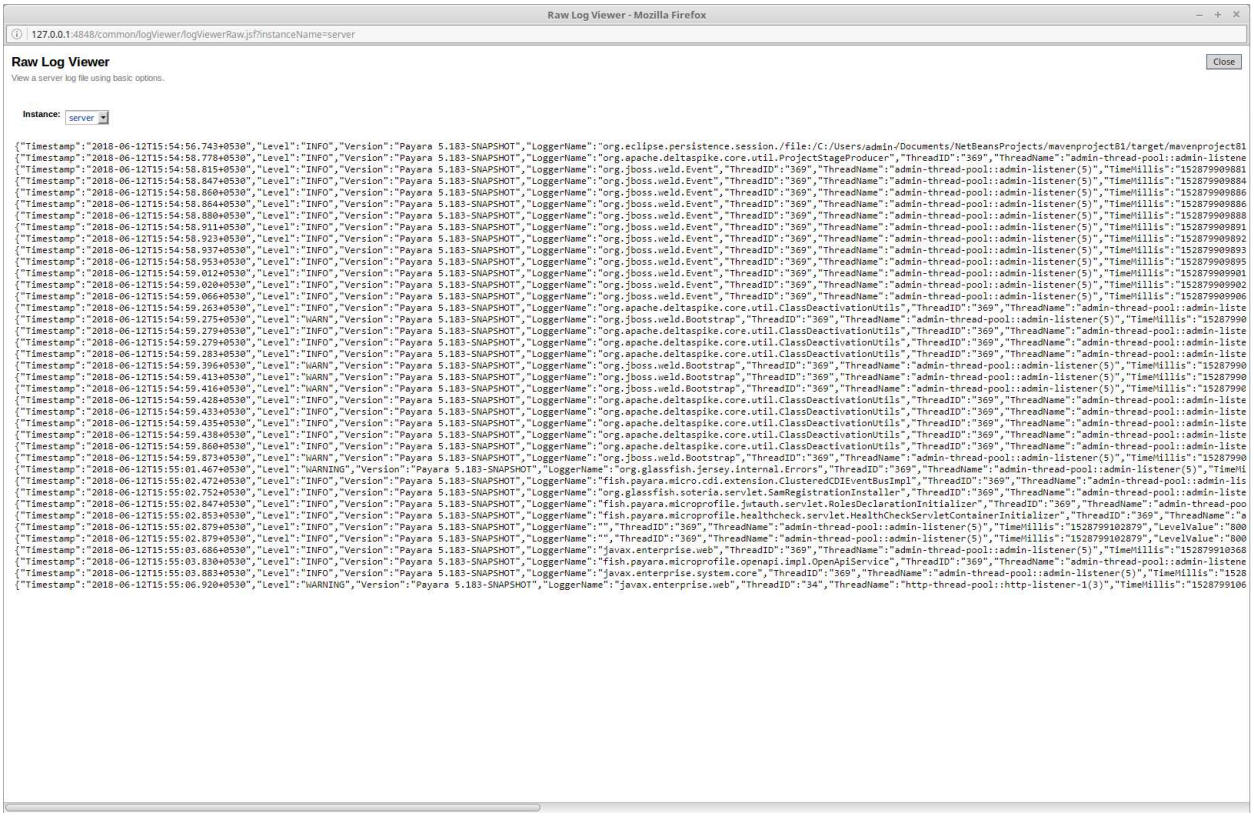
```
asadmin> set-log-attributes
com.sun.enterprise.server.logging.UniformLogFormatter.ansiColor=true
```

IMPORTANT	This setting is enabled by default when a domain is started in Linux and macOS environments.

JSON Log Format

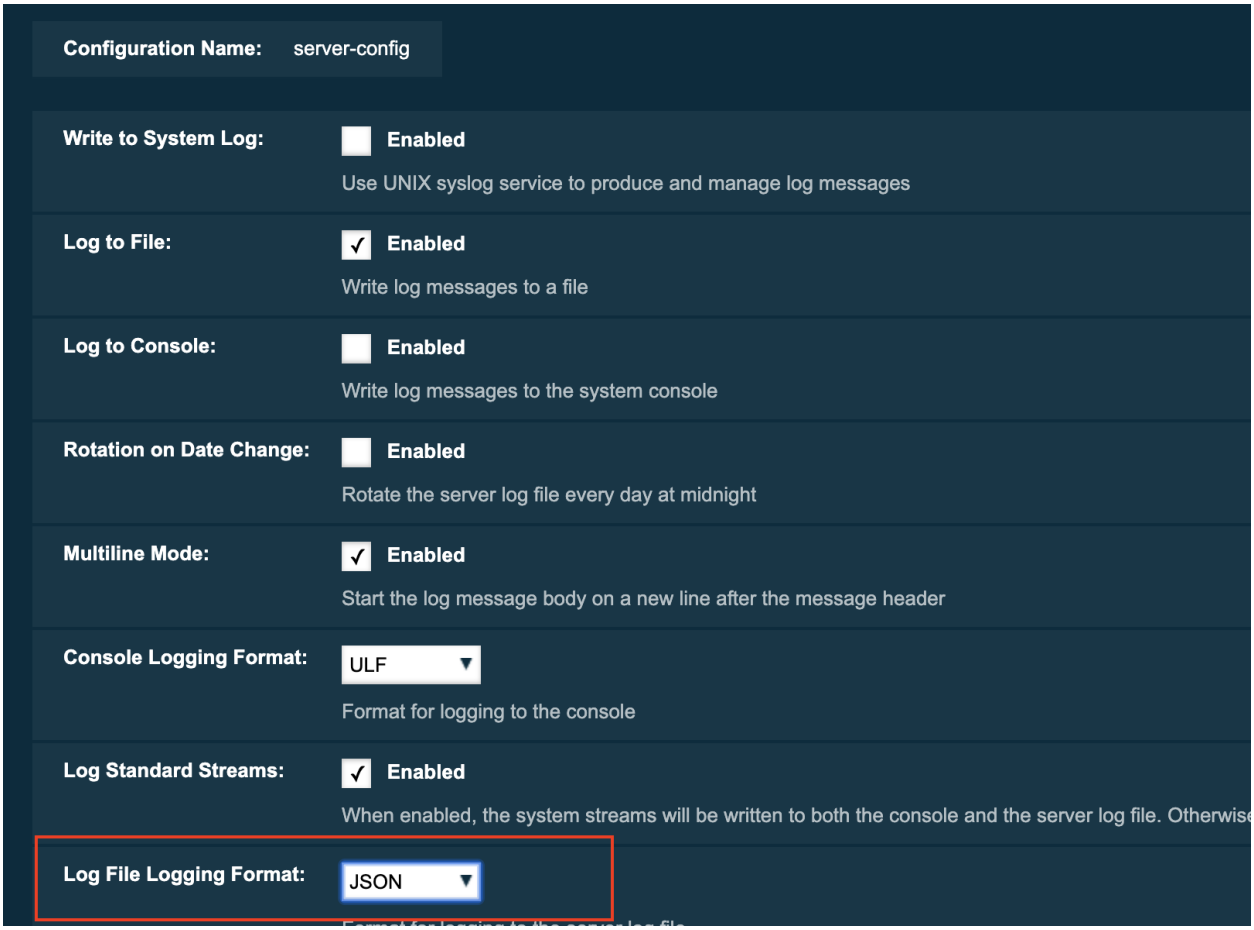
Besides the standard Uniform Log Format (ULF) and Oracle Diagnostics Logging (ODL) formats Payara Server provides a JSON format. With this format, every entry is formatted as a JSON object string. These entries can be easily processed by any JSON parser for further data processing.

Once the JSON formatter is enabled, the server's log file may look similar to this sample:



Enable the JSON Formatter Using the Web Admin Console

To enable the JSON formatter using the Admin Console, just select `JSON` from the list of Logging Formats, either for `Console` or `Log File`:



Enable the JSON Formatter Using the Asadmin CLI

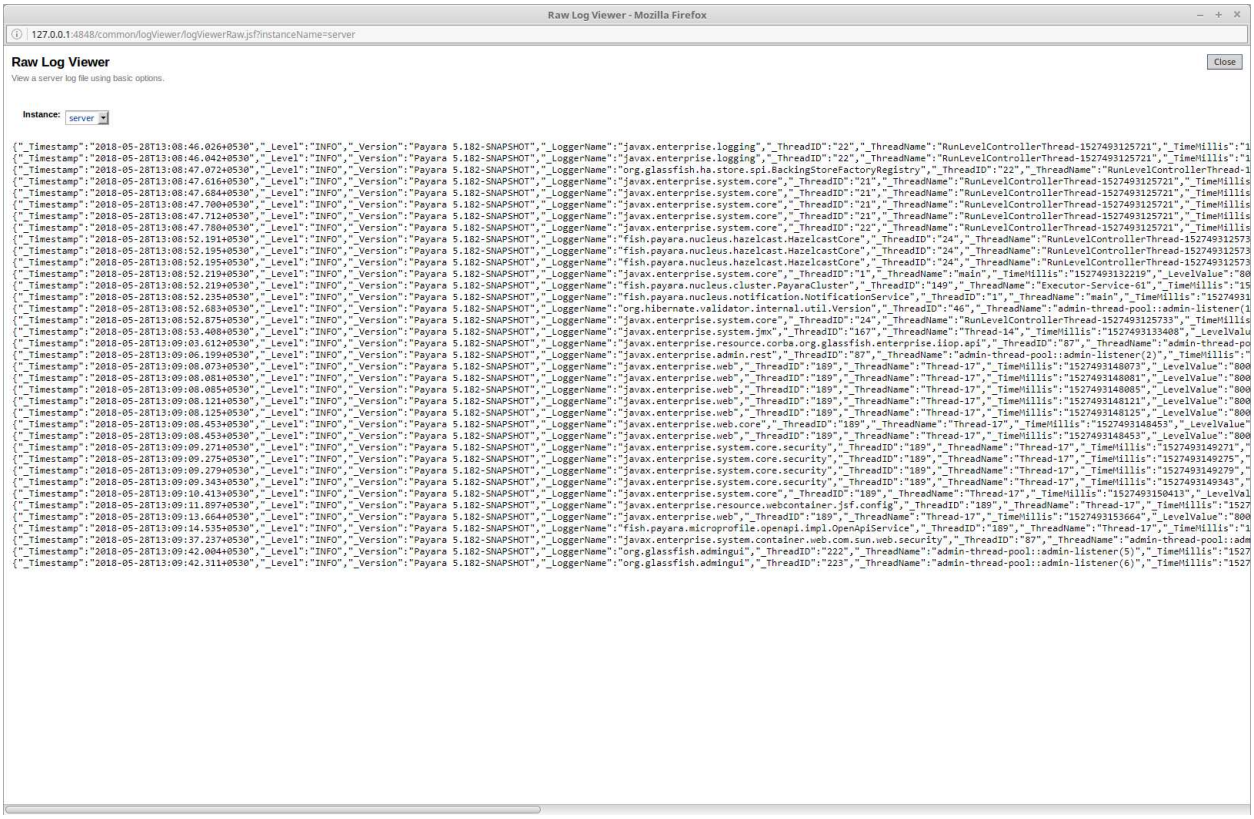
The following command will enable the JSON formatter:

```
asadmin> set-log-attributes
com.sun.enterprise.server.logging.GFFileHandler.formatter='fish.payara.enterprise.server.logging.JSONLogFormatter'
```

Configure Prefixed Field Names

In some situations, the JSON representation of a log entry may use field names that clash with existing standard field names that logging gathering tools may use them for specific purposes.

To solve this problem, Payara Server can be configured to automatically prefix all field names in the JSON object representation with an underscore (`_`) character. See the following sample to get an idea of how such a JSON payload would look like:



The following Asadmin command will enable this configuration setting:

```
asadmin set-log-attributes
fish.payara.deprecated.jsonlogformatter.underscoreprefix=true
```

Support for Additional Fields

The JSON Log Formatter also supports the customization of additional fields through the `setParameters` method of the LogRecord class. This action is done when logging a new entry at runtime, so it is limited to an application business logic context.

Here's a quick example of how to pass additional fields to the resulting JSON object by using a map with a single entry:

```
public class ApplicationComponent{

    public void businessMethod(){
        var lr = new LogRecord(Level.INFO, "Sample message");
        lr.setParameters(new Object[]{Collections.singletonMap("key", "value")});
        logger.log(lr);
    }
}
```

To Exclude Fields in Log Records

Use the `set-log-attributes` subcommand in remote mode to exclude specific name-value fields from log records. If the `excludeFields` attribute is not specified, all name-value fields are included.

The following fields can be excluded:

- `tid`
- `userId`
- `ecid`
- `timeMillis`
- `levelVal`
- `version`

Follow these instructions:

1. Ensure that the target server or cluster is running.

   Remote commands require a running server.

2. Exclude fields by using the "`set-log-attributes`" subcommand, specifying the following attribute and the fields to exclude: `com.sun.enterprise.server.logging.GFFileHandler.excludeFields=fields` where `fields` is a comma-separated list of the name-value fields to exclude from the log file.
3. To apply your change, restart Payara Server.

Example - to Exclude Fields in Log Records

This example excludes the `userId` and `levelVal` name-value fields in log records for standalone instance `ManagedServer1`:

```
asadmin set-log-attributes --target ManagedServer1
com.sun.enterprise.server.logging.GFFileHandler.excludeFields=userId,levelVal
```

```
com.sun.enterprise.server.logging.GFFileHandler.excludeFields logging attribute
value set to userId,levelVal.
The logging attributes are saved successfully for ManagedServer1-config.

Command set-log-attributes executed successfully.
```

You can view the full syntax and options of the subcommand by typing `asadmin help set-log-attributes` at the command line.

To Set Multiline Mode

Use the `set-log-attributes` command in remote mode to set multiline mode. When multiline mode is enabled (the default), the body of a log message starts on a new line after the message header and is indented.

1. Ensure that the target server or cluster is running.

   Remote commands require a running server.

2. Set multiline mode by using the "`set-log-attributes`" subcommand, specifying the following attribute and its value (`true` or `false`):
   `com.sun.enterprise.server.logging.GFFileHandler.multiLineMode=value`

3. To apply your change, restart Payara Server.

Example - to Set Multiline Mode

Multiline mode is enabled by default. The following example disables multiline mode in log files for standalone instance `ManagedServer1`:

```
asadmin set-log-attributes --target ManagedServer1

com.sun.enterprise.server.logging.GFFileHandler.multiLineMode=false
com.sun.enterprise.server.logging.GFFileHandler.multiLineMode logging attribute
value set to false.
The logging attributes are saved successfully for ManagedServer1-config.

Command set-log-attributes executed successfully.
```

You can view the full syntax and options of the subcommand by typing `asadmin help set-log-attributes` at the command line.

Setting Log File Rotation

As explained in log-files, Payara Server by default rotates log files when they reach `2 MB` in size.

However, you can change the default rotation settings. For example, you can change the file size at which the server rotates the log file, or you can configure a server to rotate log files based on a time interval.

In addition to changing when rotation occurs, you can also:

- Specify the maximum number of rotated files that can accumulate.

  By default, Payara Server does not limit the number of rotated log files that are retained. However, you can set a limit.

  After the number of log files reaches this limit, subsequent file rotations delete the oldest rotated log file.

- Rotate the log file manually.

  A manual rotation forces the immediate rotation of the target log file.

Changing the default log rotation settings is done using the `set-log-attributes` subcommand, and rotating log files manually is done using the `rotate-log` subcommand.

To Change the Rotation File Size

Use the `set-log-attributes` subcommand in remote mode to change the log rotation file size. The default target of this subcommand is the DAS.

Optionally, you can target a configuration, server, instance, or cluster. The minimum size that can be set is `500 KB`.

1. Ensure that the target server or cluster is running.
2. Change the rotation file size limit by using the "`set-log-attributes`" subcommand, specifying the following attribute and the desired limit in bytes:

`com.sun.enterprise.server.logging.GFFileHandler.rotationLimitInBytes=bytes`

3. To apply your change, restart Payara Server.

Example - to Change the Rotation File Size

The following example sets the log file rotation size to 1 MB for the standalone instance `ManagedServer1`:

```
asadmin set-log-attributes --target ManagedServer1
com.sun.enterprise.server.logging.GFFileHandler.rotationLimitInBytes=1000000

com.sun.enterprise.server.logging.GFFileHandler.rotationLimitInBytes
logging attribute set with value 1000000.
These logging attributes are set for ManagedServer1.
ManagedServer1 :
com.sun.enterprise.server.logging.GFFileHandler.rotationLimitInBytes
logging attribute set with value 1000000.
These logging attributes are set for ManagedServer1.

Command set-log-attributes executed successfully.
```

You can view the full syntax and options of the subcommand by typing `asadmin help set-log-attributes` at the command line.

To Change the File Rotation Interval

Use the `set-log-attributes` subcommand in remote mode to change the log file rotation time limit interval. The default target of this subcommand is the DAS. Optionally, you can target a configuration, server, instance, or cluster. The default value is `0`.

1. Ensure that the target server or cluster is running.
2. Change the rotation time limit by using the "`set-log-attributes`" subcommand, specifying the following attribute and the desired limit in minutes:
   `com.sun.enterprise.server.logging.GFFileHandler.rotationTimelimitInMinutes=minutes`
3. To apply your change, restart Payara Server.

Example - to Change the File Rotation Interval

The following example sets the log file rotation time limit for the cluster `Cluster1`, which has the instances `ClusterServer1` and `ClusterServer2`.

```
asadmin set-log-attributes --target Cluster1
com.sun.enterprise.server.logging.GFFileHandler.rotationTimelimitInMinutes=10

com.sun.enterprise.server.logging.GFFileHandler.rotationTimelimitInMinutes
logging attribute set with value 10.
These logging attributes are set for Cluster1.
ClusterServer1 :
com.sun.enterprise.server.logging.GFFileHandler.rotationTimelimitInMinutes
logging attribute set with value 10.
These logging attributes are set for Cluster1.

ClusterServer2 :
com.sun.enterprise.server.logging.GFFileHandler.rotationTimelimitInMinutes
logging attribute set with value 10.
These logging attributes are set for Cluster1.

Command set-log-attributes executed successfully.
```

You can view the full syntax and options of the subcommand by typing `asadmin help set-log-attributes` at the command line.

To Change the Limit Number of Retained Files

Use the `set-log-attributes` subcommand in remote mode to change the limit on the number of log files that the server creates to store old log messages. The default target of this subcommand is the DAS.Optionally, you can target a configuration, server, instance, or cluster. The default limit value is `0`, which results in no limit placed on the number of rotated log files that are retained.

1. Ensure that the target server or cluster is running.
2. Change the limit number of retained log files by using the "`set-log-attributes`" subcommand, specifying the following attribute and the desired file limit number: `com.sun.enterprise.server.logging.GFFileHandler.maxHistoryFiles=minutes`

The behavior of the `com.sun.enterprise.server.logging.GFFileHandler.maxHistoryFiles`

attribute is as follows:

- If the property is not set, Payara Server keeps a maximum of 10 rotated log files.
- If the property is set to an invalid number or null, Payara Server keeps a maximum of 10 rotated log files.
- If the property is set to 0, Payara Server retains all rotated log files (that is, sets no maximum).

1. To apply your change, restart Payara Server.

Example - to Change the Limit Number of Retained Files

The following example sets the log limit number of retained log files for the DAS to `10`.

```
asadmin set-log-attributes

com.sun.enterprise.server.logging.GFFileHandler.maxHistoryFiles=10
com.sun.enterprise.server.logging.GFFileHandler.maxHistoryFiles
logging attribute set with value 10.
These logging attributes are set for server.
Command set-log-attributes executed successfully.
```

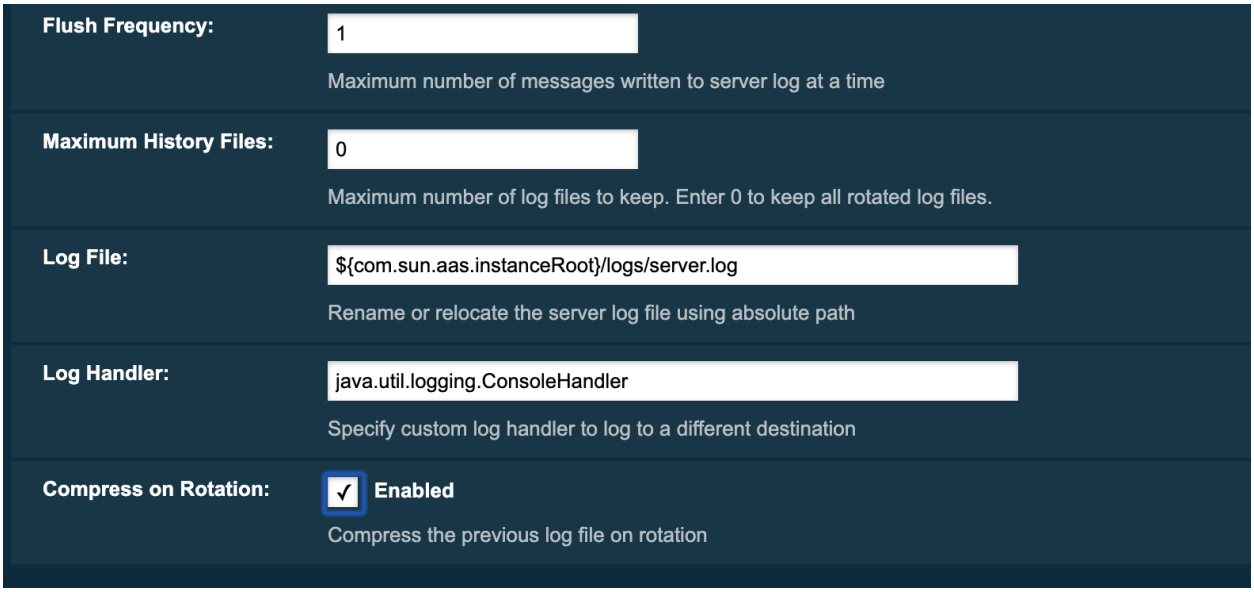
You can view the full syntax and options of the subcommand by typing `asadmin help set-log-attributes` at the command line.

Log File Compression on Rotation

Payara Server can be configured to automatically compress rotated log files in an automatic manner to save disk space.

Using the Web Admin Console

When log rotation is enabled, you can turn on automatic compression in the `Logger Settings` section of the Admin Console, by ticking the `Compress on Rotation` checkbox:



Using the Asadmin CLI

Use the following command to enable or disabled the automatic compression of log files on rotation:

```
asadmin> set-log-attributes
com.sun.enterprise.server.logging.GFFileHandler.compressOnRotation='true'
```

To Rotate Log Files Manually

You can rotate log files manually by using the `rotate-log` subcommand in remote mode. The default target of this subcommand is the DAS. Optionally, you can target a configuration, server, instance, or cluster. When you use this subcommand, the target log file is immediately moved to a new time-stamped file and a new log file is created.

Because log rotation is a dynamic operation, you do not need to restart Payara Server for changes to take effect.

1. Ensure that the target server or cluster is running.
2. Rotate log files by using the "`rotate-log`" subcommand.

Example - to Rotate Log Files Manually

The following example rotates the `server.log` file for `ManagedServer2` to `server.log_`yyyy-mm-dd`T`hh-mm-ss`, where `yyyy-mm-dd`T`hh-mm-ss`

represents the time when the file is rotated, and creates a new `server.log` file in the default location.

```
asadmin> rotate-log --target ManagedServer2
Command rotate-log executed successfully.
```

You can view the full syntax and options of the subcommand by typing `asadmin help rotate-log` at the command line.

Adding a Custom Logging Handler

By default, Payara Server log records are captured in a server log file using the format described in viewing-log-records.

However, you may find that you want to log messages to a different location, such as a database or a remote server, or log messages from specific loggers to your own file. This can be done by implementing a custom log handler.

This section explains how to add a custom log handler to the Payara Server logging service.

To Add a Custom Log Handler

A comma-separated list of log handlers is installed during startup of the Java Virtual Machine (JVM) host. The default log handler that is provided in the `logging.properties` file, `ConsoleHandler`, is configured as follows:

```
handlers=java.util.logging.ConsoleHandler
```

In Payara Server, the best approach to developing a custom handler is to define a Hundred-Kilobyte Kernel (HK2) component that implements the handler contract. Payara Server registers this handler automatically because it is an HK2 component. There is no task required of the administrator.

NOTE	The custom handler class should be packaged in an OSGi module and the JAR file placed in the `as-install/modules` directory.

To configure a custom handler that is not developed as an HK2 component, add the new handler to the `logging.properties` file after the developer has put the custom handler JAR file into the `domain-dir/lib/ext` directory.

Before You Begin

If you set a handler by setting the handlers attribute in the logging properties file, the class that extends `java.util.logging.Handler` must be in the server classpath.

1. Ensure that the target server or cluster is running. Remote subcommands require a running server.
2. Use the "`set-log-attributes`" subcommand to add the handler to the `handlers` attribute.

   The default target of this subcommand is the DAS. Optionally you can target a configuration, server, instance, or cluster.

3. To apply your changes, restart Payara Server.

Example - to Add a Custom Log Handler

This example adds the custom logger `com.example.logging.MyHandler` to the logging properties file of the DAS.

```
asadmin set-log-attributes handlers
=java.util.logging.ConsoleHandler,com.example.logging.MyHandler

handlers logging attribute set with value
java.util.logging.ConsoleHandler,com.example.logging.MyHandler.
These logging attributes are set for server.

Command set-log-attributes executed successfully.
```

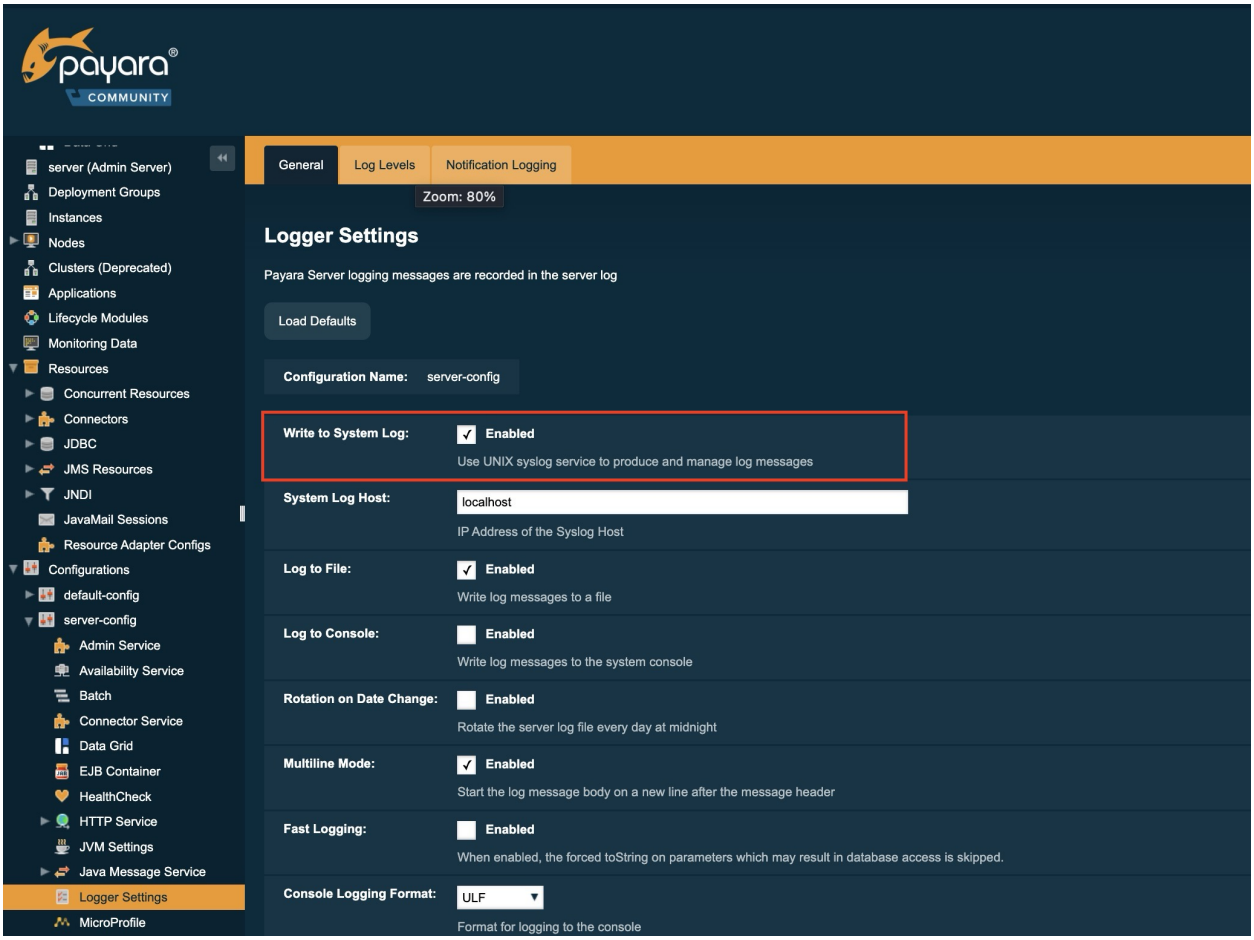
You can view the full syntax and options of the subcommand by typing `asadmin help set-log-attributes` at the command line.

Write to System Log

The **Write to System Log** option allows you to enable and disable the action of logging to the system log. Disabling this option should help to minimize disk usage.

Enable Write to System Log Using the Admin Console

To configure the `Write to System Log` option using the Admin Console:



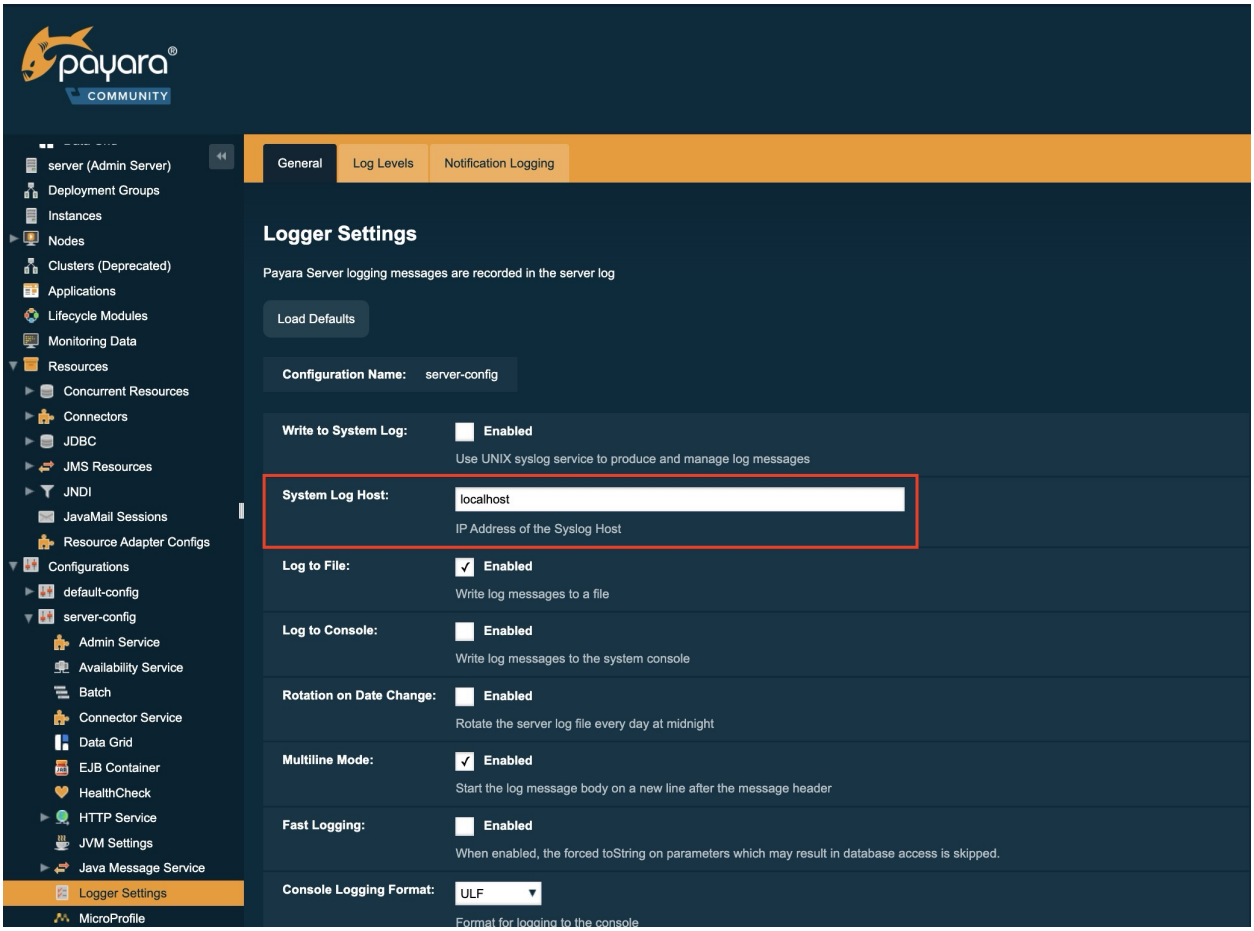
Enable Write to System Log Using the Asadmin CLI

Use the following command to enable or disable the Write to System Log feature:

```
asadmin set-log-attributes
com.sun.enterprise.server.logging.SyslogHandler.useSystemLogging=true
```

Set the System Log Host Using the Admin Console

To configure the `System Log Host` option using the Admin Console:



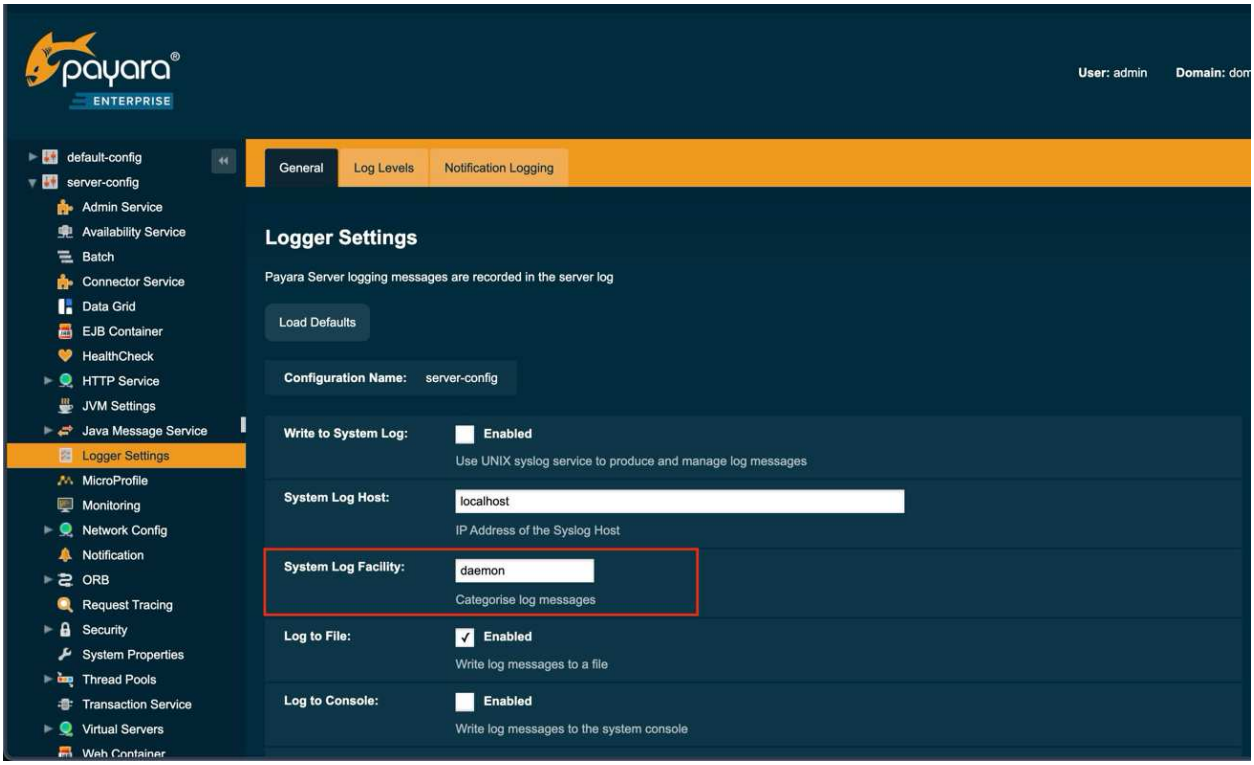
Set the System Log Host Using the Asadmin CLI

Use the following command to set the system log host:

```
asadmin set-log-attributes com.sun.enterprise.server.logging.SyslogHandler.host
=localhost
```

Set the System Log Facility Using the Admin Console

To configure the `System Log Facility` option using the Admin Console:



Set the System Log Facility Using the Asadmin CLI

Use the following command to set the system log facility:

```
asadmin set-log-attributes
com.sun.enterprise.server.logging.SyslogHandler.facility=daemon
```

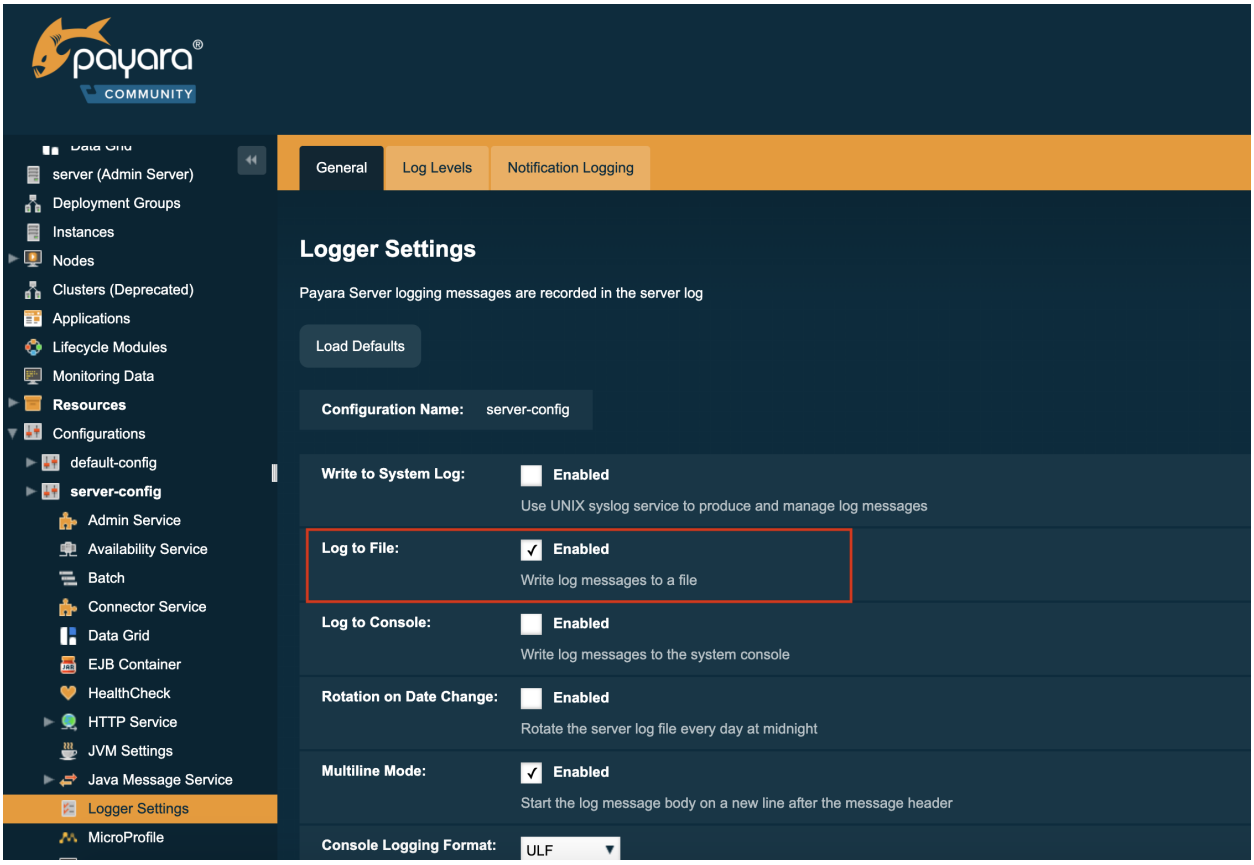
Log to File

The **Log to File** option allow you to enable and disable the action of logging to a file. When disabled this should help to minimize disk usage.

TIP	This option will be more accommodating if Payara Server is running in cloud environment where disk space is scarce.

Enable Log to File Using the Admin Console

To configure the `Log to File` option using theAdmin Console:



Enable Log to File Using the Asadmin CLI

Use the following command to enable or disable the logging to a file feature:

```
asadmin set-log-attributes
com.sun.enterprise.server.logging.GFFileHandler.logtoFile=false
```

Log to Console

The `Log to Console` option controls if the server writes the logging entries directly to the console. The property only has an effect when the server or instance is started in verbose mode.

To enable the log to console option you simply have to start the domain or instance using the `verbose` option:

```
asadmin start-domain --verbose
```

When the server or domain is not started in verbose mode, the

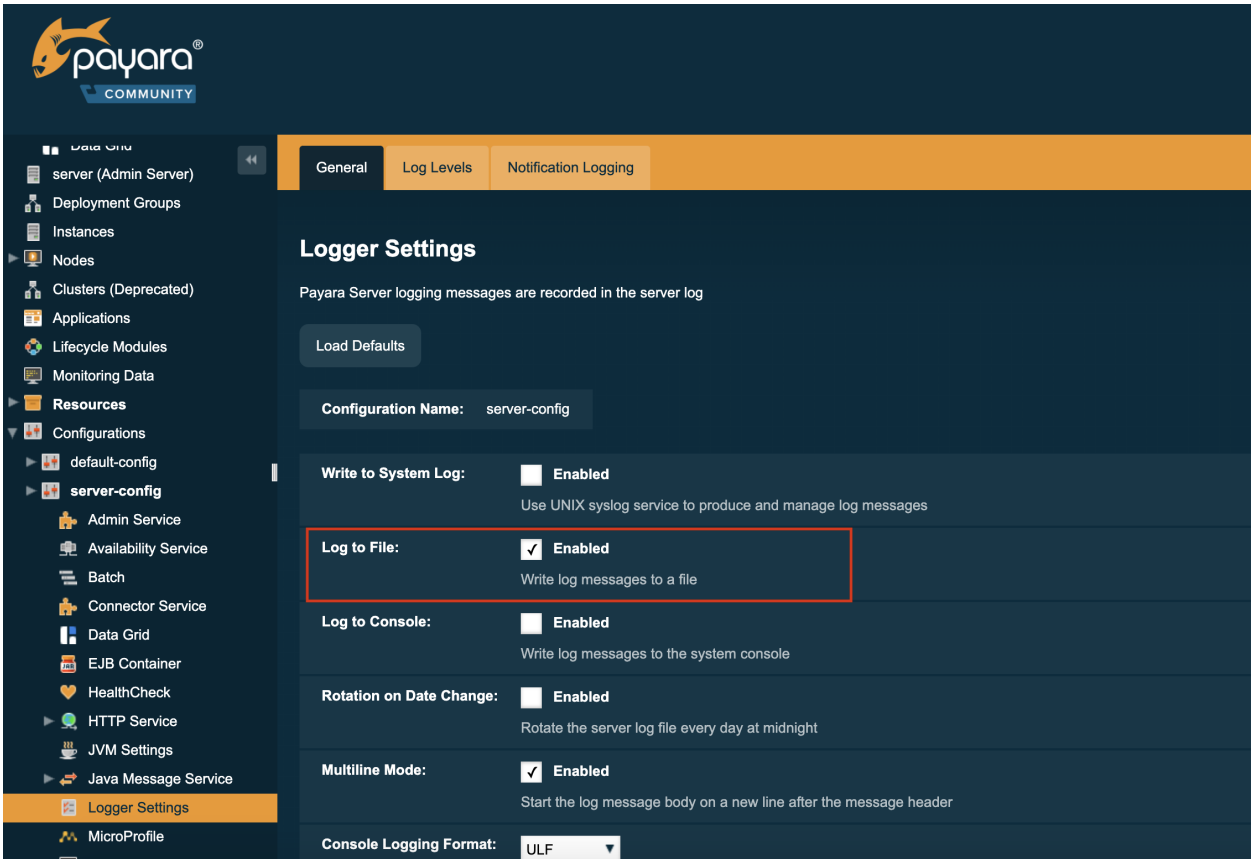
`java.util.ConsoleHandler` will not be active for logging, although it is defined as handler in the `logging.properties` file because there is no console attached with the corresponding server process.

> **NOTE** When in verbose mode, the **Log to Console** option will also be active, regardless of the initial value set in the `logging.properties` file as otherwise, no log output would be visible on the console.

Once the server is run in verbose mode, you can use either the Admin Console or the Asadmin CLI to modify this setting.

Configure Logging to Console Using the Admin Console

To configure the `Log to Console` option using the Admin Console:



Configure Logging to Console Using the Asadmin CLI

Use the following command to enable or disable the log to console feature:

```
asadmin> set-log-attributes
com.sun.enterprise.server.logging.GFFileHandler.logtoConsole=false
```

Fast Logging

Whenever logging occurs on an application, if the `LogRecord` in question contains set parameters, they will undergo a forced transformation by having a `toString()` method call.

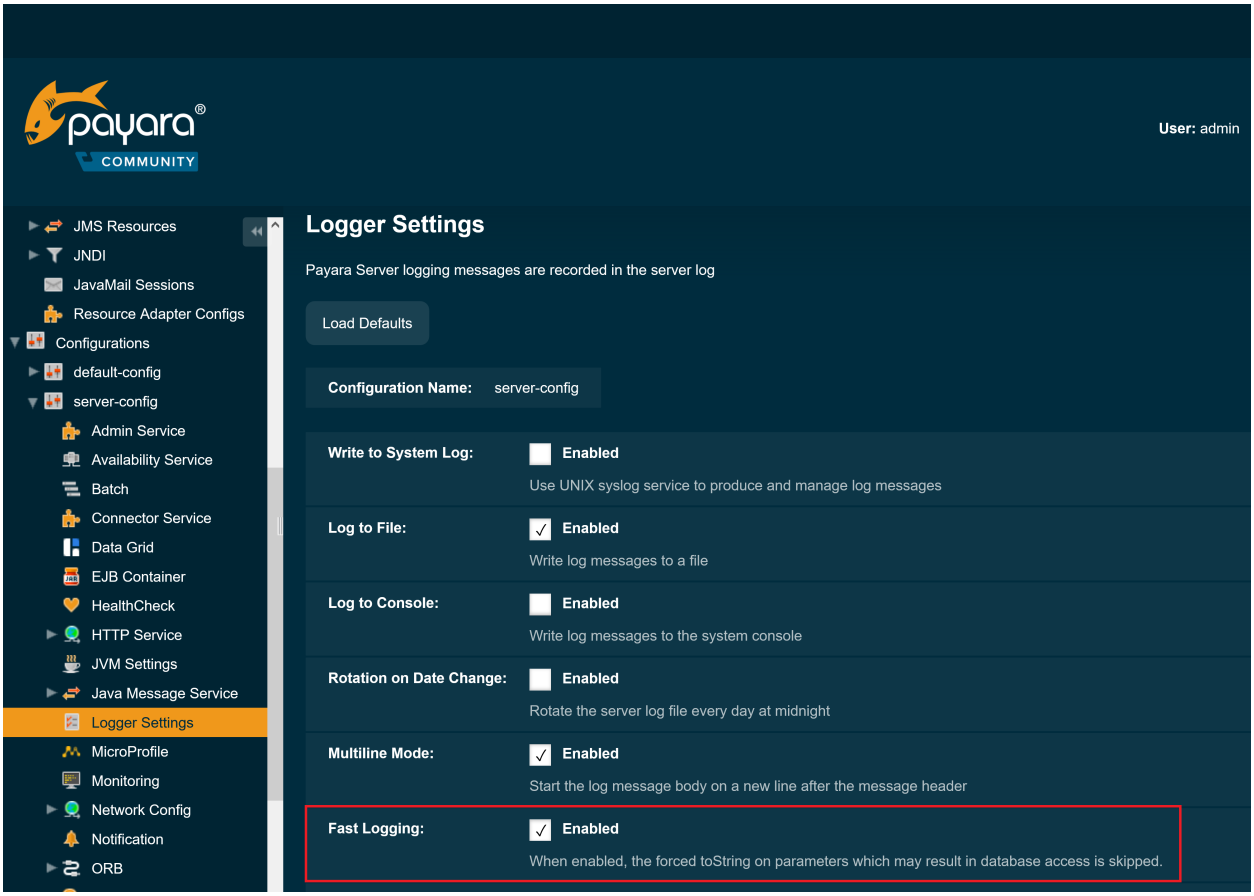
In most cases this is the desirable outcome, but will not provide the best performance. You can enable the **Fast Logging** setting to skip this forced parameter transformation at runtime.

A common use case of this feature would be to prevent database access done by JPA entities, as it is a common occurrence for entity data to be logged out for auditing purposes.

CAUTION	Enabling this setting can cause log entries that contain mutable parameters to be out of sync when they are resolved. Exert caution when using fast logging, as it may cause stale information to be printed out to the log.

Using the Admin Console

To configure the `Fast Logging` option, set it in the Logger settings.



Using Asadmin CLI

Use the following command to enable or disable fast logging:

```
asadmin set-log-attributes
com.sun.enterprise.server.logging.GFFileHandler.fastLogging=true
```

Environment Variable Support

The `logging.properties` configuration file supports environment variable substitution like in this example:

```
.level=${ENV=logLevel}
```

Whenever the server starts up or the logging properties are updated, the value for the `.level` property will be taken from the environment variable `logLevel`.

Viewing Log Records

The recommended means for general viewing of logging information is to use the Log

Viewer in the Administration Console. The Log Viewer simplifies reading, searching, and filtering log file contents. For instructions, see the Administration Console online help.

Payara Server also allows you to collect log files into a ZIP archive, which provides the means to obtain and view log files for an instance or cluster even when it is not currently running.

The following section explains how to collect all available log files for an instance or cluster and compile them into a single ZIP archive, which is done by using the `collect-log-files` subcommand.

To Collect Log Files into a ZIP Archive

Use the `collect-log-files` subcommand in remote mode to collect log files into a ZIP archive. The default target of this subcommand is the DAS. Optionally you can target a configuration, server, instance, or cluster.

1. Ensure that the target server or cluster is running.

   Remote subcommands require a running server.

2. Use the "`collect-log-files`" subcommand to create the ZIP archive.

   The default location in which the ZIP archive is created is the `domain-dir/collected-logs` directory. The `collect-log-files` subcommand allows you to specify a non-default directory in which the ZIP archive is to be created by using the `--retrieve` option set to `true`, followed by the directory name.

   The name of the ZIP file contains the timestamp, as follows: `log_`yyyy-mm-dd_hh-min-sec`.zip`

Example - to Collect Log Files into a .zip Archive

This example shows collecting the log files for the cluster `MyCluster` and compiling them into a ZIP archive in the `/space/output` directory.

```
asadmin collect-log-files --target MyCluster --retrieve true /space/output
```

```
Log files are downloaded for ClusterServer1.
Log files are downloaded for ClusterServer2.
Created Zip file under /space/output/log_2011-02-10_13-35-25.zip.

Command collect-log-files executed successfully.
```

When the ZIP file created by the preceding command is uncompressed, the following directory structure is created:

```
as-install-parent/
       glassfish/
            domains/
                domain-dir/
                       collected_logs/
                                    logs/
                                       ClusterServer1/
                                                   server.log
                                       ClusterServer2/
                                                   server.log
```

You can view the full syntax and options of the subcommand by typing `asadmin help collect-log-files` at the command line.

Listing Loggers

You can list and view information about all public loggers in your distribution of Payara Server.

To List Loggers

Use the `list-loggers` subcommand in remote mode to list the logger name, subsystem, and description of loggers in your distribution of Payara Server. Internal loggers are not listed.

1. Ensure that the target server or cluster is running.

   Remote commands require a running server.

2. List loggers by using the "`list-loggers`" subcommand.

Example - to List Loggers

This example lists the logger name, subsystem, and description for each logger. Some lines of output are omitted from this example for readability.

```
asadmin list-loggers

Logger Name                                Subsystem         Logger Description
...
jakarta.enterprise.monitoring              Monitoring        Monitoring Logger
jakarta.enterprise.system.core.ee          AS-CORE           Jakarta EE Core Kernel
jakarta.enterprise.system.jmx              JMX               JMX System Logger
jakarta.enterprise.system.tools.admin      ADMIN             Administration Services
...
Command list-loggers executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-loggers` at the command line.

Access Logging Max File Size

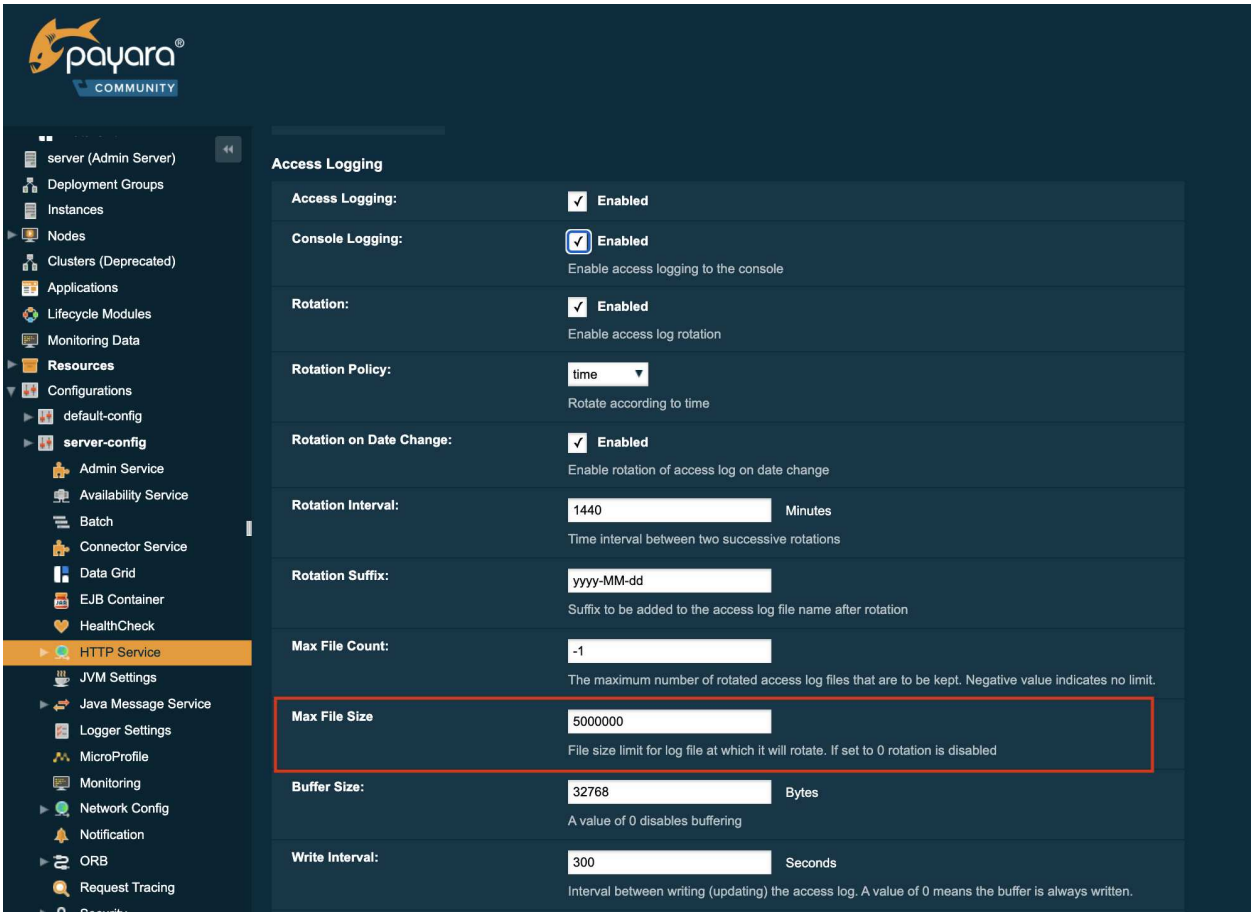
Payara Server provides different ways to rotate HTTP access log files. This section will detail the use of the max size of the log file to trigger a rotation.

The **Max File Size** option provides a way to change the file size at which the server rotates the access log file. This option accepts an integer value specifying the maximum size of the log file, after which a file rotation will occur.

IMPORTANT	If the value for this option is set to `0`, HTTP access log rotation will be disabled.

Using the Admin Console

To configure the **Max File Size** option using the Admin Console:



Using the Asadmin CLI

Use the following command to set maximum file size option:

```
asadmin> set configs.config.server-config.http-service.access-log.maximum-file-size=5000000
```

For more information on how to configure HTTP Access logging, see administering-http-configurations.

Notification Service Logging

The Notification Service captures information about events which come from other services, such as the JMX Monitoring Service, the HealthCheck service or the Request Tracing service and captures this information so that notifiers can redirect these events. The Log Notifier, turned on by default, will print these notifications events in the server's log.

It is possible to configure the Log Notifier to store its output in a separate log file.

NOTE

If this setting is enabled, the default log file will be found at `domain-dir/logs/notification.log`

The Notification Logging Service uses its own collection of logging properties which are separate from the standard logging facilities of Payara Server. However, they are stored in the same configuration file.

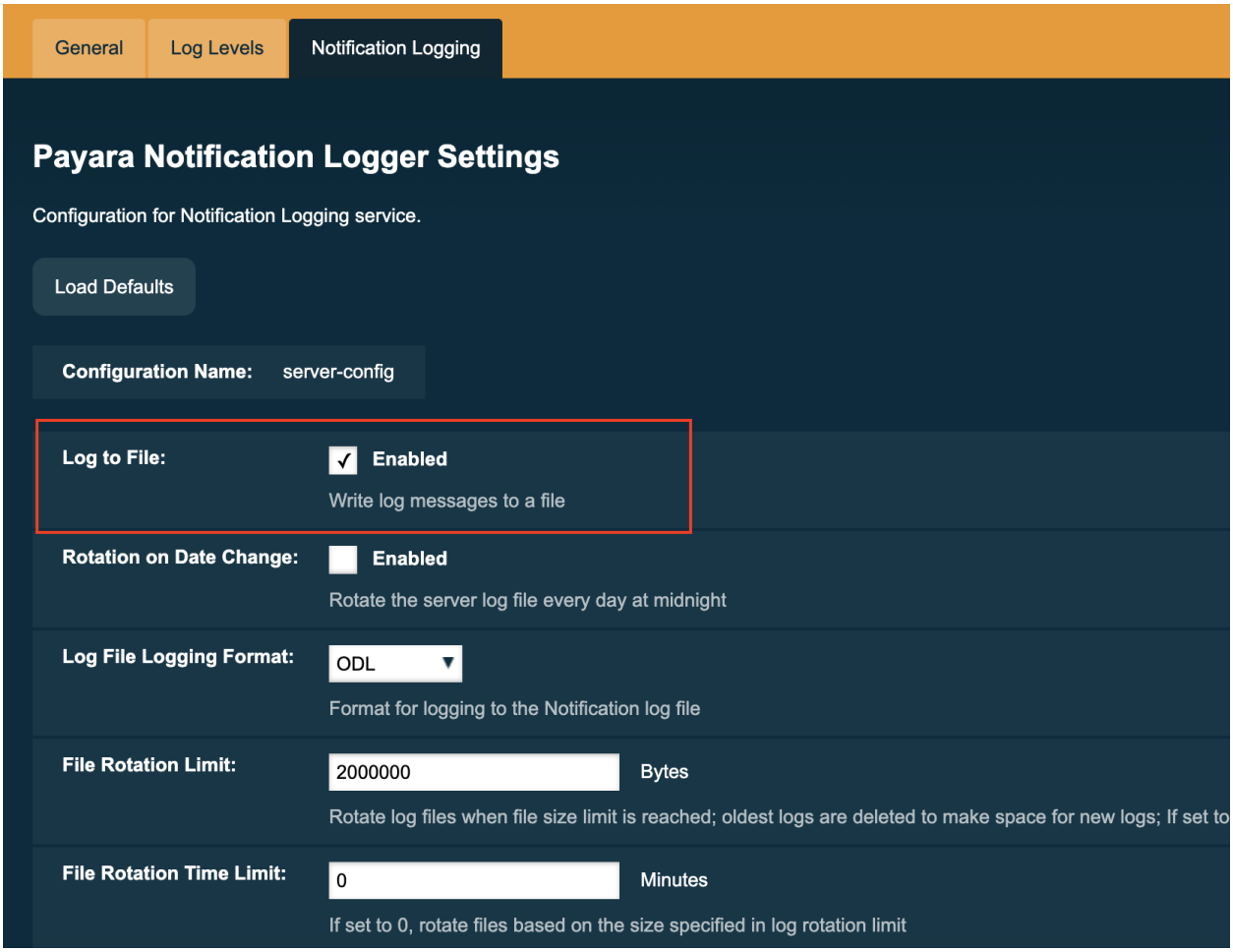
Configuring the Notification Logging Service

Enabling or Disabling Logging to a File

The **Log to File** option will provide a way to enable and disable logging to a file of Payara Server Notifications. When disabled this should help to minimize disk usage. This option will be more accommodating if Payara Server is running in cloud environment where disk space is scarce.

Using the Admin Console

To configure the **Log to File** option using Admin Console:



Using the Asadmin CLI

You use the following Asadmin command to enable/disable the logging to a file setting:

```
asadmin set-log-attributes
fish.payara.enterprise.server.logging.PayaraNotificationFileHandler.logtoFile=false
```

Configuring Log File Rotation

By default, The notification log rotation is set to rotate when the log file reaches 2 Megabytes(MB). Nonetheless, there are number of ways log rotation can be configured.

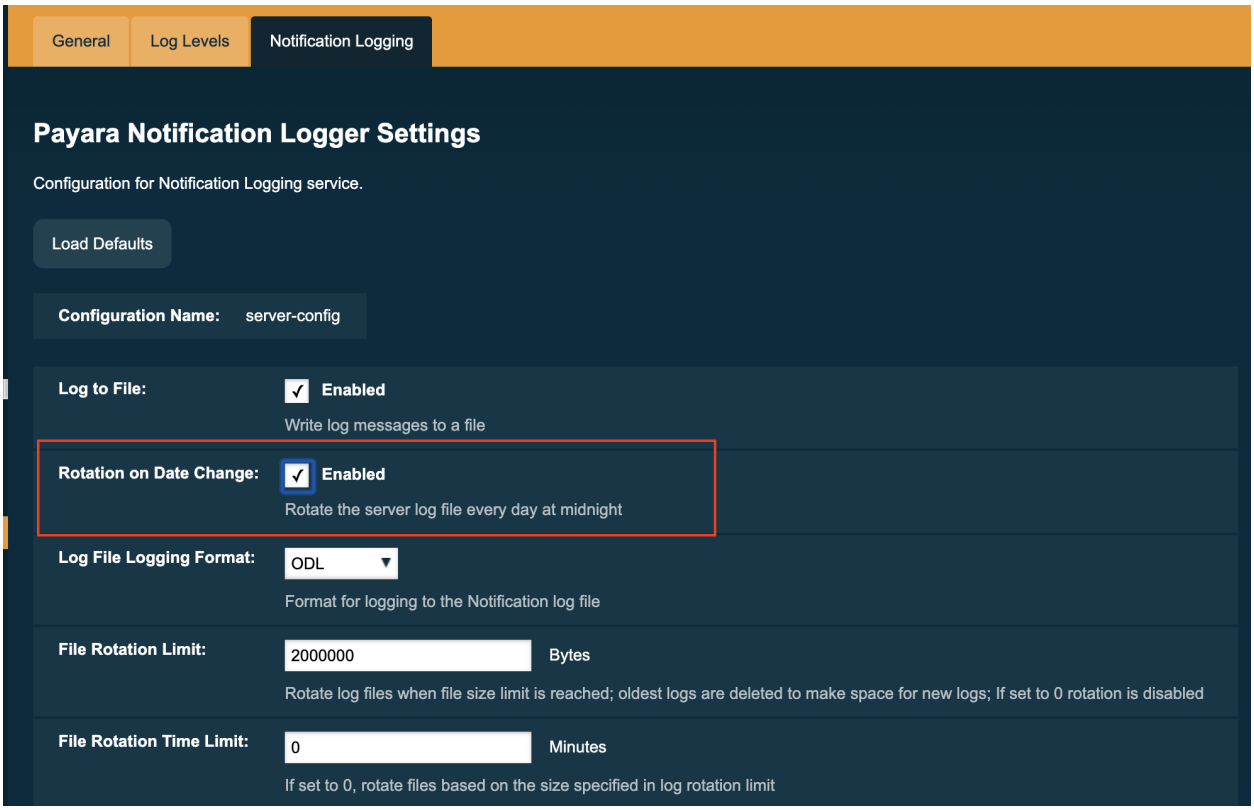
Size of file at which the log rotates can be changed, it can be set to rotate at date change, rotate when time limit is reached and, finally based on maximum number of history files.

Rotation on Date Change

The **Rotation On Date Change** option provides a way to set the log rotation when the system date changes (at midnight, by default).

Using the Admin Console

To configure the **Rotation on Date Change** option using the Admin Console:



Using the Asadmin CLI

Use the following Asadmin command to enable or disabled the log rotation on date change setting:

```
asadmin set-log-attributes
fish.payara.enterprise.server.logging.PayaraNotificationFileHandler.rotationOnDateC
hange=false
```

Rotation on File Size

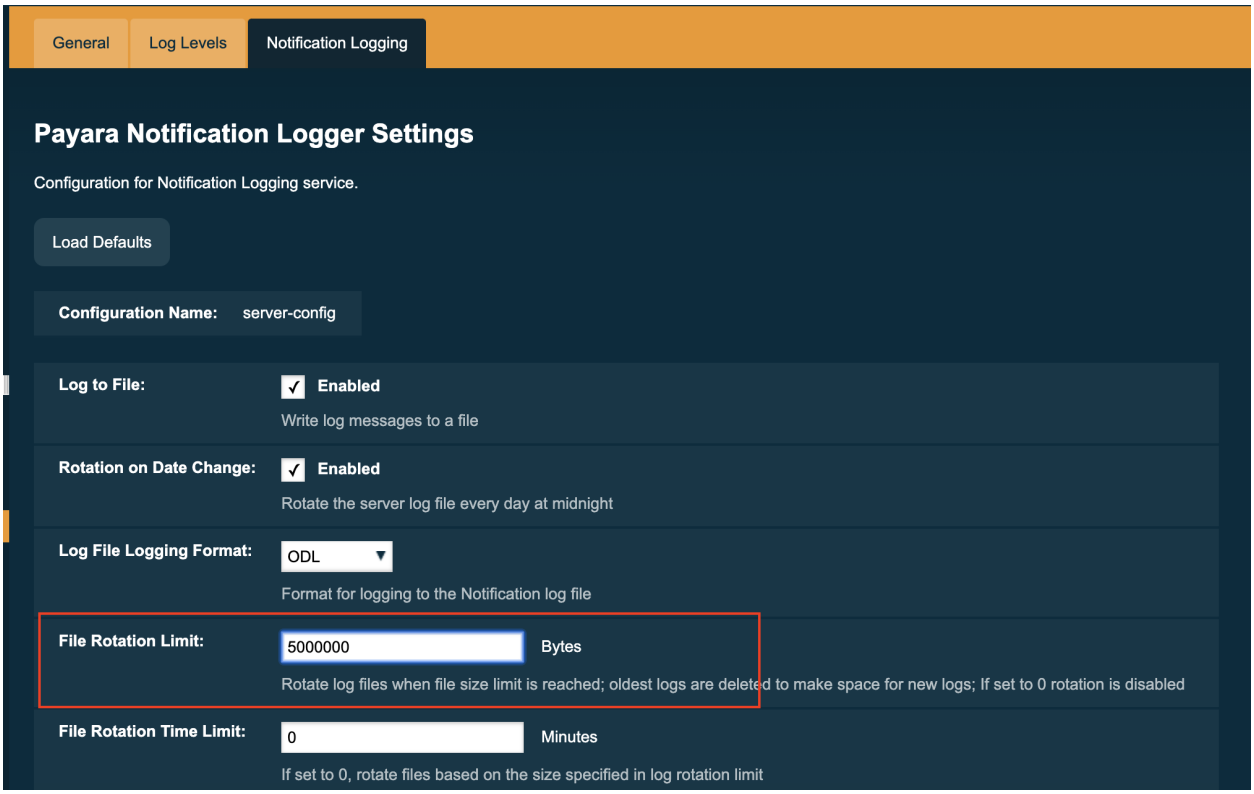
The **File Rotation Limit** option provides a way to change the file size at which the server triggers the log file rotation. This option accepts an integer value specifying the maximum size of the log file, after which a file rotation will occur.

The minimum size it can be set to is `500KB` (`500.000` bytes).

IMPORTANT	If the value for this option is set to `0`, log rotation will be disabled.

Using the Admin Console

To configure the **File Rotation Limit** option using the Admin Console:



Using the Asadmin CLI

Use the following Asadmin command to set the file rotation limit:

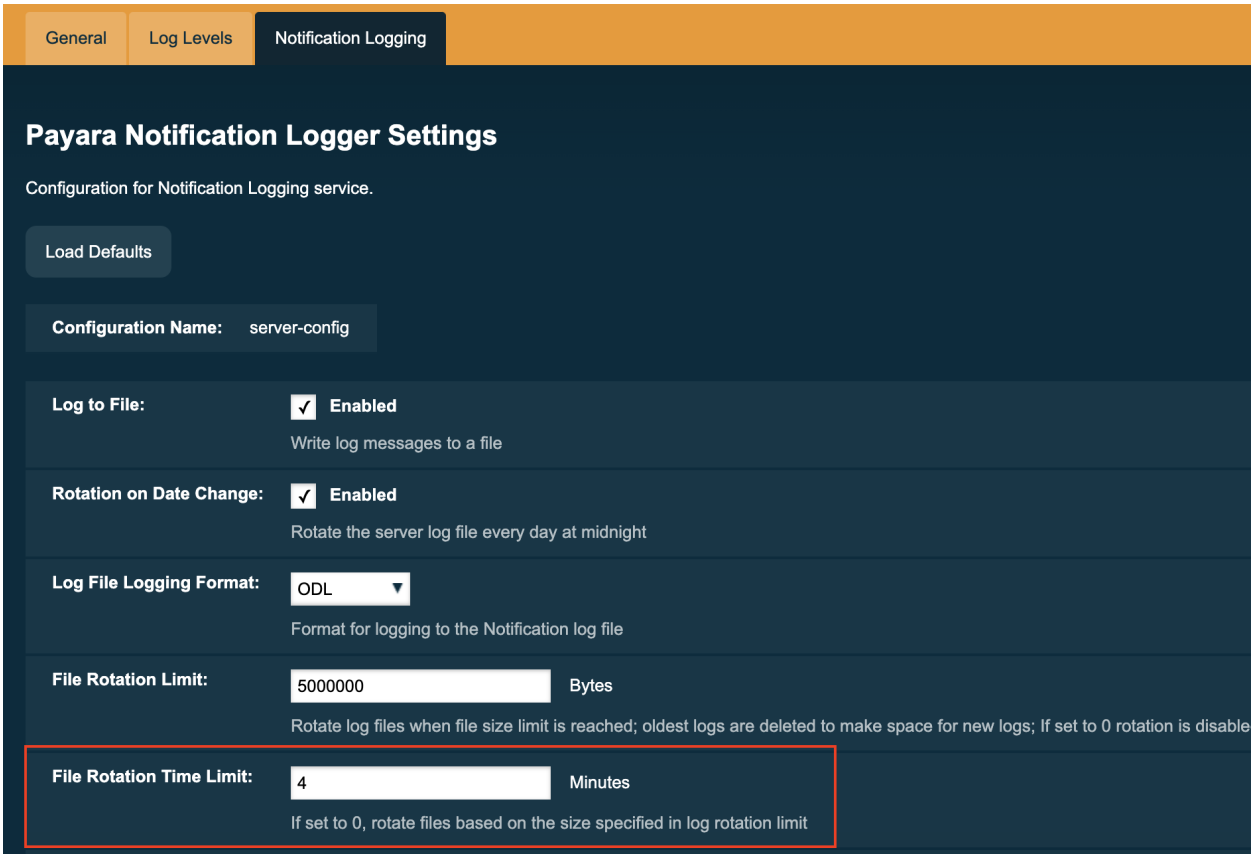
```
asadmin set-log-attributes
fish.payara.enterprise.server.logging.PayaraNotificationFileHandler.rotationLimitIn
Bytes=500000
```

Rotation on Time Limit

The **File Rotation Time Limit** option provides a way to trigger the notification log file rotation based on a fixed time limit. The value of this setting is an integer that defines the time limit in minutes until the log rotation gets triggered.

Using the Admin Console*

To configure the **File Rotation Time Limit** option using the Admin Console:



Using the Asadmin CLI

Use the following command to set the fixed time limit used to trigger the log rotation:

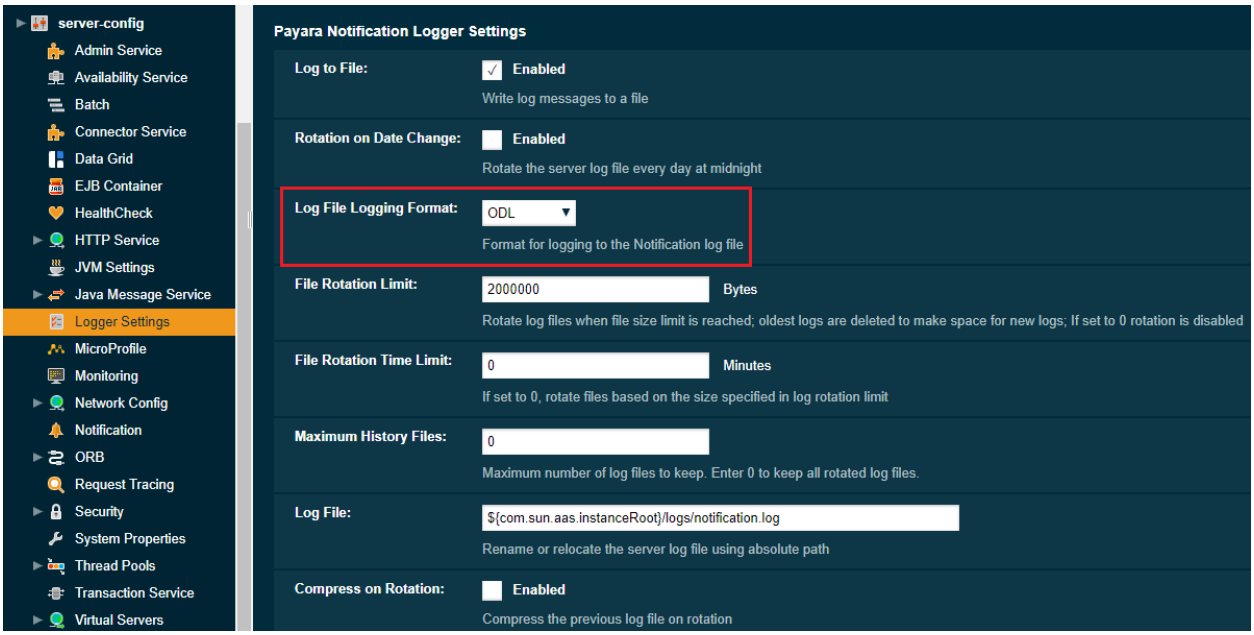
```
asadmin set-log-attributes
fish.payara.enterprise.server.logging.PayaraNotificationFileHandler.rotationTimelim
itInMinutes=4
```

Change the Logging Format

The **Log File Logging Format** option can be used to change the notification log entries' format. There are 3 logging formats available: `ULF`, `ODL` and `JSON`, each one represented by a specific formatter class present in the Azul Payara API.

Using the Admin Console

To configure the **Log File Logging Format** option using the Admin Console:



Using the Asadmin CLI

Use the following command to change the logging format:

```
asadmin set-log-attributes
fish.payara.enterprise.server.logging.PayaraNotificationFileHandler.formatter=fish.
payara.enterprise.server.logging.JSONLogFormatter
```

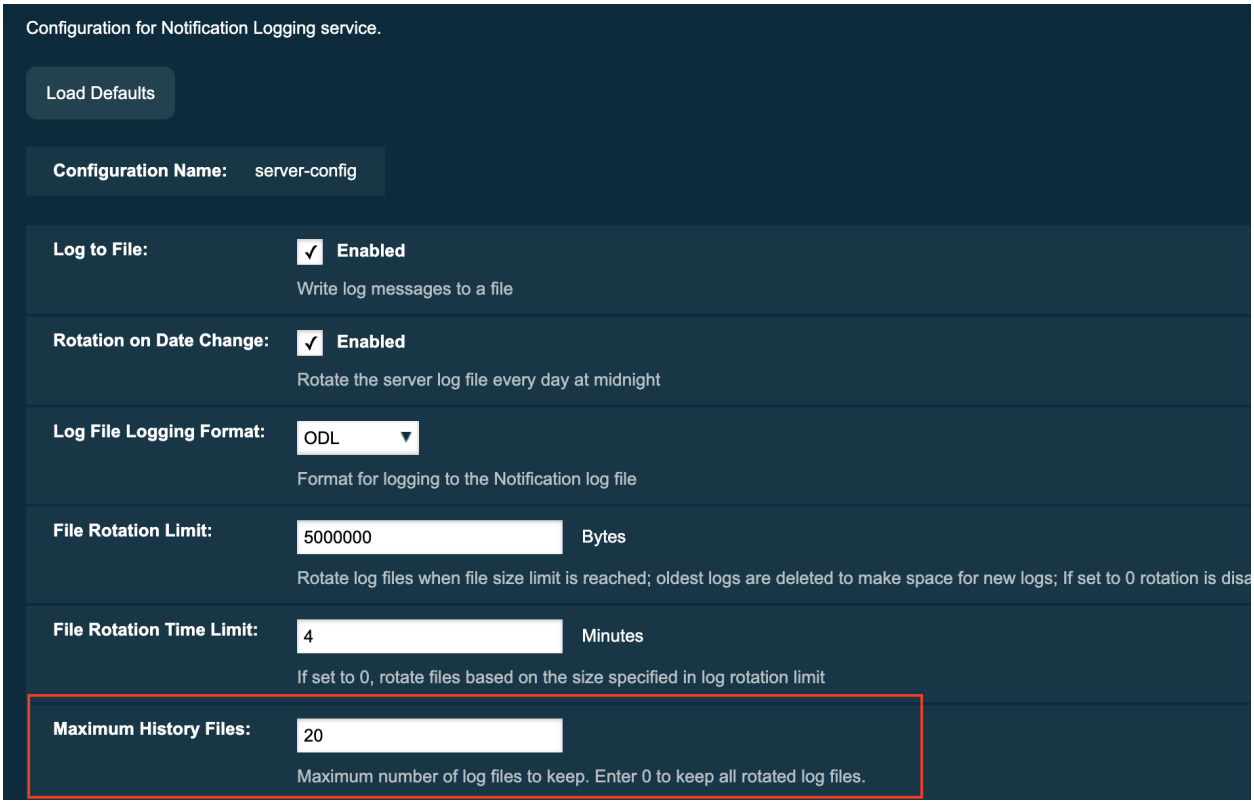
Set the Maximum Number of Historic Files

The `Maximum History Files` option provides a way to set the limit on the number of notification log files that can be created by Payara Server.

Once the number of files reaches the set limit, the oldest rotated log file is deleted. If the value for this option is set `0`, all the rotated log files are preserved.

Using the Admin Console

To configure the `Maximum History File` option using the Admin Console:



Using the Asadmin CLI

Use the following command to set the limit on the number of log files that can be created by Payara Server:

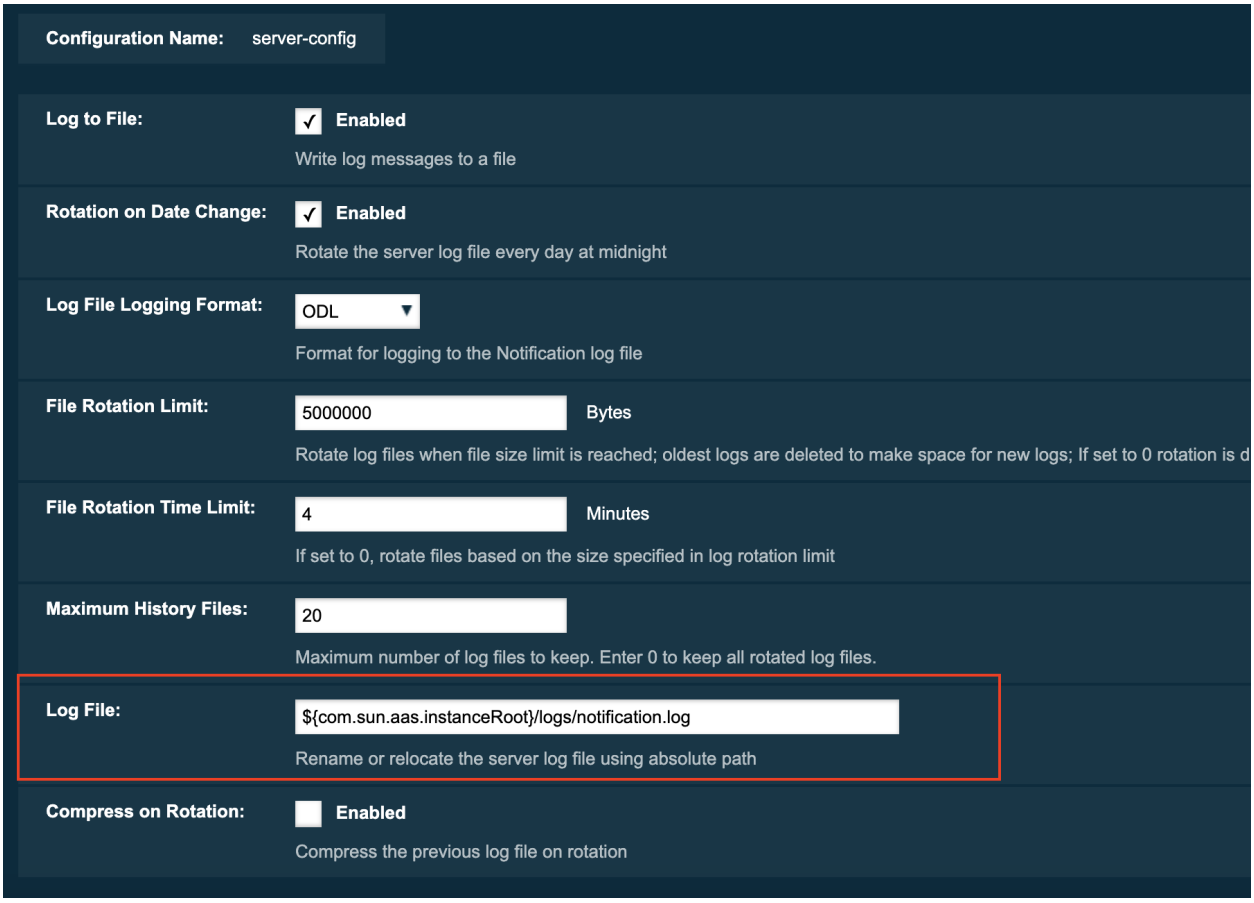
```
asadmin set-log-attributes
fish.payara.enterprise.server.logging.PayaraNotificationFileHandler.maxHistoryFiles
=20
```

Change the Name and Location of the Log File

The `Log File` option provides a way to change the default name and location of the notification log files.

Using the Admin Console

To configure `Log File` option using Admin Console:



Using the Asadmin CLI

Use the following command to change the default name and location of the server log files:

```
asadmin set-log-attributes
fish.payara.enterprise.server.logging.PayaraNotificationFileHandler.file=${com.sun.
aas.instanceRoot}/notification/testNotification.log
```

TIP

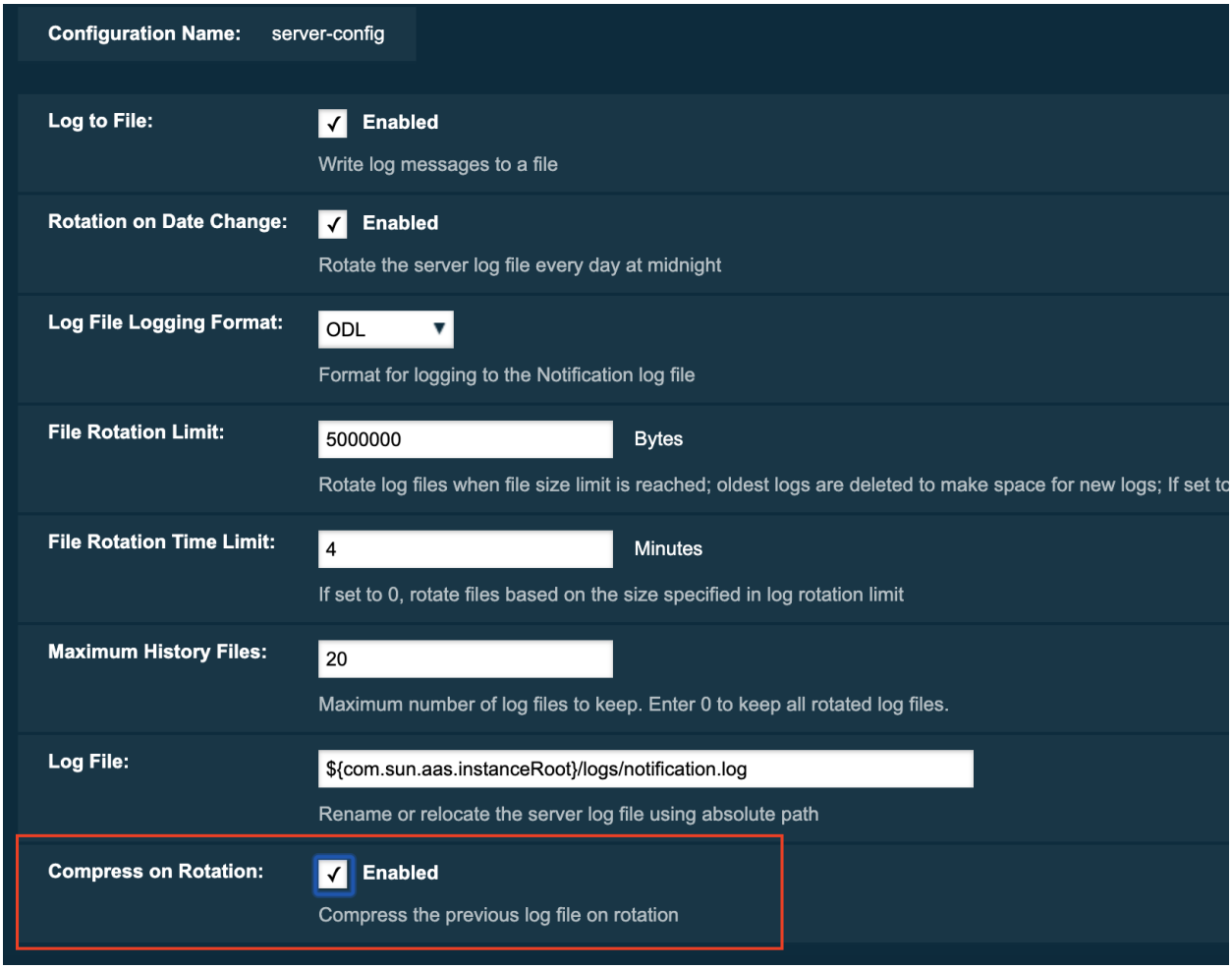
As seen in the sample command, you can use the `${com.sun.aas.instanceRoot}` system property to reference the current domain directory location.

Enable File Compression on Rotation

The `Compress on Rotation` option provides a way to enable the automatic compression of notification log files on rotation.

Using the Admin Console

To configure the `Compress on Rotation` option using the Admin Console:



Using the Asadmin CLI

Use the following command to enable or disable compression of log files on rotation:

```
asadmin> set-log-attributes
fish.payara.enterprise.server.logging.PayaraNotificationFileHandler.compressOnRotat
ion=true
```

Administering the Monitoring Service

This chapter explains how to monitor the Payara Server components and services by using the `asadmin` command-line utility. Instructions for configuring JConsole to monitor Payara Server resources are also provided.

Instructions for monitoring by using the Administration Console are contained in the

Administration Console online help.

About Monitoring

Monitoring is the process of reviewing the statistics of a system to improve performance or solve problems. The monitoring service can track and display operational statistics, such as the number of requests per second, the average response time, and the throughput. By monitoring the state of various components and services deployed in Payara Server, you can identify performance bottlenecks, predict failures, perform root cause analysis, and ensure that everything is functioning as expected.Data gathered by monitoring can also be useful in performance tuning and capacity planning.

For this release of Payara Server, monitoring is exposed in a modular way so that many client modules can access and display the monitoring statistics. These clients include the Administration Console, the `asadmin` utility, AMX, and REST interfaces.

How the Monitoring Tree Structure Works

A monitorable object is a component, subcomponent, or service that can be monitored. Payara Server uses a tree structure to track monitorable objects. Because the tree is dynamic, the tree changes as Payara Server components are added or removed.

In the tree, a monitorable object can have child objects (nodes) that represent exactly what can be monitored for that object. All child objects are addressed using the dot (.) character as a separator. These constructed names are referred to as dotted names. Detailed information on dotted names is available in the "`dotted-names`(5ASC)" help page.

The following command lists the monitorable child objects of the instance `server`:

```
asadmin> list --monitor "server.*"

server.applications
server.connector-service
server.http-service
server.jms-service
server.jvm
server.network
server.orb
server.resources
server.security
```

```
server.thread-pool
server.transaction-service
server.web
```

Each object is represented by a dotted name. Dotted names can also address specific attributes in monitorable objects. For example, the `jvm` object has a `memory` attribute with a statistic called `maxheapsize`. The following dotted name addresses the attribute:

```
server.jvm.memory.maxheapsize
```

Although an object is monitorable, it is not necessarily being actively monitored. For instructions on activating monitoring, see configuring-monitoring.

Tree Structure of Monitorable Objects

Each monitorable object has a hierarchical tree structure. In the tree, a replaceable such as *statistics represents the name of the attribute that you can show statistics for.

Applications Tree Hierarchy

The `applications` tree contains the following nodes:

```
server.applications
    |--- application1
    |     |--- ejb-module-1
    |     |         |--- ejb1 *
    |     |                   |--- bean-cache (for entity/sfsb) *
    |     |                   |--- bean-pool (for slsb/mdb/entity) *
    |     |                   |--- bean-methods
    |     |                        |---method1 *
    |     |                        |---method2 *
    |     |                   |--- timers (for s1sb/entity/mdb) *
    |     |--- web-module-1
    |     |         |--- virtual-server-1 *
    |     |                        |---servlet1  *
    |     |                        |---servlet2  *
    |--- standalone-web-module-1
    |     |         |----- virtual-server-2 *
    |     |                        |---servlet3 *
    |     |                        |---servlet4 *
    |     |         |----- virtual-server-3 *
    |     |                        |---servlet3 *(same servlet on different vs)
    |     |                        |---servlet5 *
    |--- standalone-ejb-module-1
    |     |         |--- ejb2 *
    |     |                   |--- bean-cache (for entity/sfsb) *
    |     |                   |--- bean-pool (for slsb/mdb/entity) *
```

```
    |    |                    |--- bean-methods
    |    |                         |--- method1 *
    |    |                         |--- method2 *
    |    |                    |--- timers (for s1sb/entity/mdb) *
    |--- jersey-application-1
    |    |--- jersey
    |    |        |--- resources
                           resource-0
                               hitcount
                                    *statistic
    |--- application2
```

An example dotted name might be:

```
server.applications.hello.server.request.maxtime
```

An example dotted name under the EJB `method` node might be:

```
server.applications.ejbsfapp1.ejbsfapp1ejbmod1\.jar.SFApp1EJB1
```

An example Jersey dotted name might be:

```
server.applications.helloworld-webapp.jersey.resources.resource-
0.hitcount.resourcehitcount-count
```

For available statistics, see ejb-statistics, jersey-statistics, and web-statistics.

Connector Service Tree Hierarchy

The `connector-service` tree holds monitorable attributes for pools such as the connector connection pool. The `connector-service` tree contains the following nodes:

```
server.connector-service
        resource-adapter-1
             connection-pools
                  pool-1
             work-management
```

An example dotted name might be `server.connector-service.resource-adapter-1.connection-pools.pool-1`. For available statistics, see jmsconnector-service-statistics.

HTTP Service Tree Hierarchy

The `http-service` tree contains the following nodes:

```
server.http-service
       virtual-server
           request
               *statistic
       _asadmin
           request
               *statistic
```

An example dotted name under the virutal-server node might be `server.http-service.virtual-server1.request.requestcount`. For available statistics, see http-service-statistics.

Jms/container Service Tree Hierarchy

The `jms-service` tree holds monitorable attributes for connection factories (connection pools for resource adapters) and work management (for Message Queue resource adapters). The `jms-service` tree contains the following nodes:

```
server.jms-service
        connection-factories
             connection-factory-1
        work-management
```

An example dotted name under the `connection-factories` node might be `server.jms-service.connection-factories.connection-factory-1` which shows all the statistics for this connection factory. For available statistics, see jmsconnector-service-statistics.

JVM Tree Hierarchy

The `jvm` tree contains the following nodes:

```
server.jvm
           class-loading-system
           compilation-system
           garbage-collectors
           memory
           operating-system
           runtime
```

An example dotted name under the `memory` node might be `server.jvm.memory.maxheapsize`. For available statistics, see jvm-statistics.

Network Tree Hierarchy

The network statistics apply to the network listener, such as `admin-listener`, `http-listener-1`, `ttp-listener-2`. The `network` tree contains the following nodes:

```
server.network
          type-of-listener
              keep-alive
                    *statistic
              file-cache
                    *statistic
              thread-pool
                    *statistic
              connection-queue
                     *statistic
```

An example dotted name under the `network` node might be `server.network.admin-listener.keep-alive.maxrequests-count`. For available statistics, see network-statistics.

ORB Tree Hierarchy

The `orb` tree holds monitorable attributes for connection managers. The `orb` tree contains the following nodes:

```
server.orb
    transport
        connectioncache
            inbound
                *statistic
            outbound
                *statistic
```

An example dotted name might be `server.orb.transport.connectioncache.inbound.connectionsidle-count`. For available statistics, see orb-statistics-connection-manager.

Resources Tree Hierarchy

The `resources` tree holds monitorable attributes for pools such as the JDBC connection pool and connector connection pool. The `resources` tree contains the

following nodes:

```
server.resources
       connection-pool
           request
               *statistic
```

An example dotted name might be `server.resources.jdbc-connection-pool1.numconnfree.count`. For available statistics, see resource-statistics-connection-pool.

Security Tree Hierarchy

The security tree contains the following nodes:

```
server.security
       ejb
          *statistic
       web
          *statistic
       realm
          *statistic
```

An example dotted name might be `server.security.realm.realmcount-starttime`. For available statistics, see security-statistics.

Thread Pool Tree Hierarchy

The `thread-pool` tree holds monitorable attributes for connection managers, and contains the following nodes:

```
server.thread-pool
                orb
                    threadpool
                            thread-pool-1
                                *statistic
```

An example dotted name might be `server.thread-pool.orb.threadpool.thread-pool-1.averagetimeinqueue-current`. For available statistics, see thread-pool-statistics.

Transactions Service Tree Hierarchy

The `transaction-service` tree holds monitorable attributes for the transaction subsystem for the purpose of rolling back transactions. The `transaction-service` tree contains the following nodes:

```
server.transaction-service
         statistic
```

An example dotted name might be `server.tranaction-service.activeids`. For available statistics, see transaction-service-statistics.

Web Tree Hierarchy

The `web` tree contains the following nodes:

```
server.web
           jsp
              *statistic
           servlet
              *statistic
           session
              *statistic
           request
              *statistic
```

An example dotted name for the `servlet` node might be `server.web.servlet.activeservletsloadedcount`. For available statistics, see web-module-common-statistics.

About Monitoring for Add-On Components

An add-on component typically generates statistics that Payara Server can gather at runtime. Adding monitoring capabilities enables an add-on component to provide statistics to Payara Server in the same way as components that are supplied in the Payara Server distributions. As a result, you can use the same administrative interfaces to monitor statistics from any installed Payara Server component, regardless of the origin of the component.

Tools for Monitoring Payara Server

The following `asadmin` subcommands are provided for monitoring the ervices and

components of Payara Server:

- The `enable-monitoring`, `disable-monitoring`, or the `get` and `set`subcommands are used to turn monitoring on or off. For instructions, see configuring-monitoring.
- The `monitor type` subcommand is used to display basic data for a particular type of monitorable object. For instructions, see viewing-common-monitoring-data.
- The `list --monitor` subcommand is used to display the objects that can be monitored with the `monitor` subcommand. For guidelines and instructions, see guidelines-for-using-the-list-and-get-subcommands-for-monitoring.
- The `get` subcommand is used to display comprehensive data, such as the attributes and values for a dotted name. The `get` subcommand used with a wildcard parameter displays all available attributes for any monitorable object. For additional information, see guidelines-for-using-the-list-and-get-subcommands-for-monitoring.

Configuring Monitoring

By default, the monitoring service is enabled for Payara Server, but monitoring for the individual modules is not. To enable monitoring for a module, you change the monitoring level for that module to `LOW` or `HIGH`, You can choose to leave monitoring `OFF` for objects that do not need to be monitored.

- `LOW`. Simple statistics, such as create count, byte count, and so on
- `HIGH`. Simple statistics plus method statistics, such as method count, duration, and so on
- `OFF`. No monitoring, no impact on performance

To Enable Monitoring

Use the `enable-monitoring` subcommand to enable the monitoring service itself, or to enable monitoring for individual modules. Monitoring is immediately activated, without restarting Payara Server.

You can also use the set subcommand to enable monitoring for a module. Using the

`set` command is not a dynamic procedure, so you need to restart Payara Server for your changes to take effect.

- Determine which services and components are currently enabled for monitoring.

```
asadmin> get server.monitoring-service.module-monitoring-levels.*
```

This example output shows that the HTTP service is not enabled (OFF for monitoring), but other objects are enabled:

```
configs.config.server-config.monitoring-service.module-monitoring-levels.web-
container=HIGH
       configs.config.server-config.monitoring-service.module-monitoring-
levels.http-service=OFF
           configs.config.server-config.monitoring-service.module-monitoring-
levels.jvm=HIGH
```

- Enable monitoring by using the "`enable-monitoring`" subcommand. Server restart is not required.

Example - Enable Monitoring

This example enables the monitoring service without affecting monitoring for individual modules.

```
asadmin> enable-monitoring
Command enable-monitoring executed successfully
```

Example - Configure Monitoring

This example enables the monitoring service and enables the use of MBeans (JMX Monitoring) and the AMX subsystem in one command:

```
asadmin> set-monitoring-service-configuration --enabled=true --mbeansenabled=true
--target=server
Command set-monitoring-service-configuration executed successfully
```

Example - Enable Monitoring Levels

This example enables monitoring for the HTTP service by setting the monitoring level to `HIGH` (you must restart the server for changes to take effect).

```
asadmin> set server.monitoring-service.module-monitoring-levels.http-service=HIGH
Command set executed successfully
```

An alternative to the previous command that doesn't require a server restart is the following:

```
asadmin> set-monitoring-level --module=http-service --level=HIGH --target=server
```

The `set-monitoring-level` command can be used to configure the monitoring levels of multiple modules as well:

```
asadmin> set-monitoring-level --module=jvm,http-service --level=HIGH --target
=server
```

To Disable Monitoring

Use the `disable-monitoring` subcommand to disable the monitoring service itself, or to disable monitoring for individual modules. Monitoring is immediately stopped, without restarting Payara Server.

You can also use the set-monitoring-level subcommand to disable monitoring for a module. To do this, set its `--level` option to `OFF`.

- Determine which modules are enabled for monitoring using the `get-monitoring-level` command:

```
asadmin get-monitoring-level --target=server

Module                        Monitoring Level
cloud                         OFF
cloud-elasticity              OFF
cloud-orchestrator            OFF
cloud-tenant-manager          OFF
cloud-virt-assembly-service   OFF
connector-connection-pool     OFF
connector-service             OFF
deployment                    OFF
ejb-container                 OFF
http-service                  LOW
jdbc-connection-pool          OFF
jersey                        OFF
jms-service                   OFF
jpa                           OFF
jvm                           HIGH
```

```
orb                             OFF
security                        OFF
thread-pool                     OFF
transaction-service             OFF
web-container                   OFF
web-services-container          OFF

Command get-monitoring-level executed successfully.
```

As the io

- Disable monitoring for a service or module by using the "`disable-monitoring`" subcommand.
  Server restart is not required.

Example Disable Monitoring

This example disables the monitoring service without changing the monitoring levels for individual modules.

```
asadmin> disable-monitoring
Command disable-monitoring executed successfully
```

Example Disable Monitoring for Specific Modules

This example disables monitoring for specific modules. Their monitoring levels are set to `OFF`.

```
asadmin> disable-monitoring --modules web-container,ejb-container
Command disable-monitoring executed successfully
```

Viewing Common Monitoring Data

Use the `monitor` subcommand to display basic data on commonly-monitored objects.

To View Common Monitoring Data

Use the `--type` option of the `monitor` subcommand to specify the object for which you want to display data, such as `httplistener`, `jvm`, `webmodule`. If you use the `monitor` subcommand without specifying a type, an error message is displayed.

Output from the subcommand is displayed continuously in a tabular format. The `--interval` option can be used to display output at a particular interval (the default is

30 seconds).

Before You Begin

A monitorable object must be configured for monitoring before you can display data on the object. See to-enable-monitoring.

1. Determine which type of monitorable object you want to monitor.
   Your choices for 5.0 are `jvm`, `httplistener`, and `webmodule`.
2. Request the monitoring data by using the "`monitor`" subcommand.

Example 8-6 View Common Monitoring Data

This example requests common data for type `jvm` on instance `server`.

```
asadmin> monitor --type jvm server

UpTime(ms)                          Heap and NonHeap Memory(bytes)
current                   min        max         low         high        count

9437266                   8585216    619642880   0           0           93093888
9467250                   8585216    619642880   0           0           93093888
```

You can also view the full syntax and options of the subcommand by typing `asadmin help monitor` at the command line.

Common Monitoring Statistics

*HTTP Listener Common Statistics

The statistics available for the `httplistener` type are shown in the following table.

Table 8-1 HTTP Listener Common Monitoring Statistics

Statistic	Description
`ec`	Error count. Cumulative value of the error count
`mt`	Maximum time. Longest response time for a request; not a cumulative value, but the largest response time from among the response times
`pt`	Processing time. Cumulative value of the times taken to process each request, with processing time being the average of request processing times over request

Statistic	Description
`rc`	Request count. Cumulative number of requests processed so far

JVM Common Statistics

The statistics available for the `jvm` type are shown in the following table.

Table 8-2 JVM Common Monitoring Statistics

Statistic	Description
`count`	Amount of memory (in bytes) that is guaranteed to be available for use by the JVM machine
`high`	Retained for compatibility with other releases
`low`	Retained for compatibility with other releases
`max`	The maximum amount of memory that can be used for memory management.
`min`	Initial amount of memory (in bytes) that the JVM machine requests from the operating system for memory management during startup
`UpTime`	Number of milliseconds that the JVM machine has been running since it was last started

Web Module Common Statistics

The statistics available for the `webmodule` type are shown in the following table.

Table 8-3 Web Module Common Monitoring Statistics

Statistic	Description
`ajlc`	Number of active JavaServer Pages (JSP) technology pages that are loaded
`asc`	Current active sessions
`aslc`	Number of active servlets that are loaded
`ast`	Total active sessions
`mjlc`	Maximum number of JSP pages that are loaded
`mslc`	Maximum number of servlets that are loaded
`rst`	Total rejected sessions

Statistic	Description
`st`	Total sessions
`tjlc`	Total number of JSP pages that are loaded
`tslc`	Total number of servlets that are loaded

Viewing Comprehensive Monitoring Data

By applying the `list` and `get` subcommands against the tree structure using dotted names, you can display more comprehensive monitoring data, such as a description of each of the statistics and its unit of measurement.

Guidelines for Using the `list` and `get` Subcommands for Monitoring

The underlying assumptions for using the `list` and `get` subcommands with dotted names are:

- A `list` subcommand that specifies a dotted name that is not followed by a wildcard (`*`) lists the current node's immediate children. For example, the following subcommand lists all immediate children belonging to the `server` node:

```
list --monitor server
```

- A `list` subcommand that specifies a dotted name followed by a wildcard of the form `.*` lists a hierarchical tree of child nodes from the specified node. For example, the following subcommand lists all children of the `applications` node, their subsequent child nodes, and so on:

```
list --monitor server.applications.*
```

- A `list` subcommand that specifies a dotted name preceded or followed by a wildcard of the form *dottedname or dotted * name or dottedname * lists all nodes and their children that match the regular expression created by the specified matching pattern.
- A `get` subcommand followed by a `. or a` gets the set of attributes and their values

that belong to the node specified.

For example, the following table explains the output of the `list` and `get` subcommands used with the dotted name for the `resources` node.

Table 8-4 Example Resources Level Dotted Names

Subcommand	Dotted Name	Output
`list --monitor`	`server.resources`	List of pool names.
`list --monitor`	`server.resources.``connection-pool1`	No attributes, but a message saying "Use `get` subcommand with the `--monitor` option to view this node's attributes and values."
`get --monitor`	`server.resources.``connection-pool1.*`	List of attributes and values corresponding to connection pool attributes.

For detailed information on dotted names, see the "`dotted-names`" help page.

To View Comprehensive Monitoring Data

Although the `monitor` subcommand is useful in many situations, it does not offer the complete list of all monitorable objects. To work with comprehensive data for an object type, use the `list monitor` and the`get monitor` subcommands followed by the dotted name of a monitorable object.

Before You Begin

A monitorable object must be configured for monitoring before you can display information about the object. See to-enable-monitoring if needed.

- List the objects that are enabled for monitoring by using the "`list`" subcommand. For example, the following subcommand lists all components and services that have monitoring enabled for instance `server`.

```
asadmin> list --monitor "*"
server.web
```

```
server.connector-service
server.orb
server.jms-serviceserver.jvm
server.applications
server.http-service
server.thread-pools
```

- Get data for a monitored component or service by using the "`get`" subcommand.

Example 8-7 Viewing Attributes for a Specific Type

This example gets information about all the attributes for object type `jvm` on instance `server`.

```
asadmin> get --monitor server.jvm.*
server.jvm.class-loading-system.loadedclasscount = 3715
server.jvm.class-loading-system.totalloadedclasscount = 3731
server.jvm.class-loading-system.unloadedclasscount = 16
server.jvm.compilation-system.name-current = HotSpot Client Compiler
server.jvm.compilation-system.totalcompilationtime = 769
server.jvm.garbage-collectors.Copy.collectioncount = 285
server.jvm.garbage-collectors.Copy.collectiontime = 980
server.jvm.garbage-collectors.MarkSweepCompact.collectioncount = 2
server.jvm.garbage-collectors.MarkSweepCompact.collectiontime = 383
server.jvm.memory.committedheapsize = 23498752
server.jvm.memory.committednonheapsize = 13598720
server.jvm.memory.initheapsize = 0
server.jvm.memory.initnonheapsize = 8585216
server.jvm.memory.maxheapsize = 66650112
server.jvm.memory.maxnonheapsize = 100663296
server.jvm.memory.objectpendingfinalizationcount = 0
server.jvm.memory.usedheapsize = 19741184
server.jvm.memory.usednonheapsize = 13398352
server.jvm.operating-system.arch-current = x86
server.jvm.operating-system.availableprocessors = 2
server.jvm.operating-system.name-current = Windows XP
server.jvm.operating-system.version-current = 5.1
server.jvm.runtime.classpath-current = glassfish.jar
server.jvm.runtime.inputarguments-current = []
server.jvm.runtime.managementspecversion-current = 1.0
server.jvm.runtime.name-current = 4372@ABBAGANI_WORK
server.jvm.runtime.specname-current = Java Virtual Machine Specification
server.jvm.runtime.specvendor-current = Sun Microsystems Inc.
server.jvm.runtime.specversion-current = 1.0
server.jvm.runtime.uptime = 84813
server.jvm.runtime.vmname-current = Java HotSpot(TM) Client VM
server.jvm.runtime.vmvendor-current = Sun Microsystems Inc.
server.jvm.runtime.vmversion-current = 1.5.0_11-b03
```

Example 8-8 Viewing Monitorable Applications

This example lists all the monitorable applications for instance `server`.

```
asadmin> list --monitor server.applications.*
server.applications.app1
server.applications.app2
server.applications.app1.virtual-server1
server.applications.app2.virtual-server1
```

Example 8-9 Viewing Attributes for an Application

This example gets information about all the attributes for application `hello`.

```
asadmin> get --monitor server.applications.hello.*
server.applications.hello.server.activatedsessionstotal = 0
server.applications.hello.server.activejspsloadedcount = 1
server.applications.hello.server.activeservletsloadedcount = 1
server.applications.hello.server.activesessionscurrent = 1
server.applications.hello.server.activesessionshigh = 1
server.applications.hello.server.errorcount = 0
server.applications.hello.server.expiredsessionstotal = 0
server.applications.hello.server.maxjspsloadedcount = 1
server.applications.hello.server.maxservletsloadedcount = 0
server.applications.hello.server.maxtime = 0
server.applications.hello.server.passivatedsessionstotal = 0
server.applications.hello.server.persistedsessionstotal = 0
server.applications.hello.server.processingtime = 0.0
server.applications.hello.server.rejectedsessionstotal = 0
server.applications.hello.server.requestcount = 0
server.applications.hello.server.sessionstotal =
server.applications.hello.server.totaljspsloadedcount = 0
server.applications.hello.server.totalservletsloadedcount = 0
```

Example 8-10 Viewing a Specific Attribute

This example gets information about the `jvm` attribute `runtime.vmversion-current` on instance `server`.

```
asadmin> get --monitor server.jvm.runtime.vmversion-current
server.jvm.runtime.vmversion-current = 10.0-b23
```

Comprehensive Monitoring Statistics

You can get comprehensive monitoring statistics by forming a dotted name that specifies the statistic you are looking for. For example, the following dotted name will display the cumulative number of requests for the HTTP service on `virtual-server1`:

`server.http-service.virtual-server1.request.requestcount`

EJB Statistics

EJBs fit into the tree of objects as shown in applications-tree-hierarchy. Use the following dotted name pattern to get EJB statistics for an application:

```
server.applications.appname.ejbmodulename.ejbname.bean-cache.statistic
```

NOTE

EJB statistics for an application are available after the application is executed. If the application is deployed but has not yet been executed, all counts will show default values. When the application is undeployed, all its monitoring data is lost.

EJB Cache Statistics

Use the following dotted name pattern for EJB cache statistics:

```
server.applications.appname.ejbmodulename.bean-cache.ejbname.statistic
```

The statistics available for EJB caches are listed in the following table.

Table 8-5 EJB Cache Monitoring Statistics

Statistic	Data Type	Description
`cachemisses`	RangeStatistic	The number of times a user request does not find a bean in the cache.
`cachehits`	RangeStatistic	The number of times a user request found an entry in the cache.
`numbeansincache`	RangeStatistic	The number of beans in the cache. This is the current size of the cache.
`numpassivations`	CountStatistic	Number of passivated beans. Applies only to stateful session beans.
`numpassivationerrors`	CountStatistic	Number of errors during passivation. Applies only to stateful session beans.

Statistic	Data Type	Description
`numexpiredsessionsremoved`	CountStatistic	Number of expired sessions removed by the cleanup thread. Applies only to stateful session beans.
`numpassivationsuccess`	CountStatistic	Number of times passivation completed successfully. Applies only to stateful session beans.

EJB Container Statistics

Use the following dotted name pattern for EJB container statistics:

```
server.applications.appname.ejbmodulename.container.ejbname
```

The statistics available for EJB containers are listed in the following table.

Table 8-6 EJB Container Monitoring Statistics

Statistic	Data Type	Description
`createcount`	CountStatistic	Number of times an EJB's `create` method is called.
`messagecount`	CountStatistic	Number of messages received for a message-driven bean.
`methodreadycount`	RangeStatistic	Number of stateful or stateless session beans that are in the `MethodReady` state.
`passivecount`	RangeStatistic	Number of stateful session beans that are in `Passive` state.
`pooledcount`	RangeStatistic	Number of entity beans in pooled state.
`readycount`	RangeStatistic	Number of entity beans in ready state.
`removecount`	CountStatistic	Number of times an EJB's `remove` method is called.

EJB Method Statistics

Use the following dotted name pattern for EJB method statistics:

```
server.applications.appname.ejbmodulename.bean-methods.ejbname.statistic
```

The statistics available for EJB method invocations are listed in the following table.

Table 8-7 EJB Method Monitoring Statistics

Statistic	Data Type	Description
`executiontime`	CountStatistic	Time, in milliseconds, spent executing the method for the last successful/unsuccessful attempt to run the operation. This is collected for stateless and stateful session beans and entity beans if monitoring is enabled on the EJB container.
`methodstatistic`	TimeStatistic	Number of times an operation is called; the total time that is spent during the invocation, and so on.
`totalnumerrors`	CountStatistic	Number of times the method execution resulted in an exception. This is collected for stateless and stateful session beans and entity beans if monitoring is enabled for the EJB container.
`totalnumsuccess`	CountStatistic	Number of times the method successfully executed. This is collected for stateless and stateful session beans and entity beans if monitoring enabled is true for EJB container.

EJB Pool Statistics

Use the following dotted name pattern for EJB pool statistics:

```
server.applications.appname.ejbmodulename.bean-pool.ejbname.statistic
```

The statistics available for EJB pools are listed in the following table.

Table 8-8 EJB Pool Monitoring Statistics

Statistic	Data Type	Description
`jmsmaxmessagesload`	CountStatistic	The maximum number of messages to load into a JMS session at one time for a message-driven bean to serve. Default is 1. Applies only to pools for message driven beans.
`numbeansinpool`	RangeStatistic	Number of EJBs in the associated pool, providing information about how the pool is changing.
`numthreadswaiting`	RangeStatistic	Number of threads waiting for free beans, giving an indication of possible congestion of requests.

Statistic	Data Type	Description
`totalbeanscreated`	CountStatistic	Number of beans created in associated pool since the gathering of data started.
`totalbeansdestroyed`	CountStatistic	Number of beans destroyed from associated pool since the gathering of data started.

Timer Statistics

Use the following dotted name pattern for timer statistics:

```
server.applications.appname.ejbmodulename.timers.ejbname.statistic
```

The statistics available for timers are listed in the following table.

Table 8-9 Timer Monitoring Statistics

Statistic	Data Type	Description
`numtimerscreated`	CountStatistic	Number of timers created in the system.
`numtimersdelivered`	CountStatistic	Number of timers delivered by the system.
`numtimersremoved`	CountStatistic	Number of timers removed from the system.

HTTP Service Statistics

The HTTP service fits into the tree of objects as shown in http-service-tree-hierarchy.

HTTP Service Virtual Server Statistics

Use the following dotted name pattern for HTTP service virtual server statistics:

```
server.http-service.virtual-server.request.statistic
```

The HTTP service statistics for virtual servers are shown in the following table.

Table 8-10 HTTP Service Virtual Server Monitoring Statistics

Statistic	Data Type	Description
`count200`	CountStatistic	Number of responses with a status code equal to 200

Statistic	Data Type	Description
`count2xx`	CountStatistic	Number of responses with a status code in the 2xx range
`count302`	CountStatistic	Number of responses with a status code equal to 302
`count304`	CountStatistic	Number of responses with a status code equal to 304
`count3xx`	CountStatistic	Number of responses with a status code equal in the 3xx range
`count400`	CountStatistic	Number of responses with a status code equal to 400
`count401`	CountStatistic	Number of responses with a status code equal to 401
`count403`	CountStatistic	Number of responses with a status code equal to 403
`count404`	CountStatistic	Number of responses with a status code equal to 404
`count4xx`	CountStatistic	Number of responses with a status code equal in the 4xx range
`count503`	CountStatistic	Number of responses with a status code equal to 503
`count5xx`	CountStatistic	Number of responses with a status code equal in the 5xx range
`countother`	CountStatistic	Number of responses with a status code outside the 2xx, 3xx, 4xx, and 5xx range
`errorcount`	CountStatistic	Cumulative value of the error count, with error count representing the number of cases where the response code was greater than or equal to 400
`hosts`	StringStatistic	The host (alias) names of the virtual server
`maxtime`	CountStatistic	Longest response time for a request; not a cumulative value, but the largest response time from among the response times
`processingtime`	CountStatistic	Cumulative value of the times taken to process each request, with processing time being the average of request processing times over the request count
`requestcount`	CountStatistic	Cumulative number of requests processed so far
`state`	StringStatistic	The state of the virtual server

Jersey Statistics

Jersey fits into the tree of objects as shown in applications-tree-hierarchy.

Use the following dotted name pattern for Jersey statistics:

```
server.applications.jersey-application.jersey.resources.resource-
0.hitcount.statistic
```

The statistics available for Jersey are shown in the following table.

Table 8-11 Jersey Statistics

Statistic	Data Type	Description
`resourcehitcount`	CountStatistic	Number of hits on this resource class
`rootresourcehitcount`	CountStatistic	Number of hits on this root resource class

Jms/connector Service Statistics

The JMS/Connector Service fits into the tree of objects as shown in jmscontainer-service-tree-hierarchy.

Connector Connection Pool Statistics (JMS)

Use the following dotted name pattern for JMS/Connector Service connection pool statistics:

```
server.connector-service.resource-adapter-1.connection-pool.statistic
```

JMS/Connector Service statistics available for the connector connection pools are shown in the following table.

NOTE

In order to improve system performance, connection pools are initialized lazily; that is, a pool is not initialized until an application first uses the pool or the pool is explicitly pinged. Monitoring statistics for a connection pool are not available until the pool is initialized.

Table 8-12 Connector Connection Pool Monitoring Statistics (JMS)

Statistic	Data Type	Description
`averageconnwaittime`	CountStatistic	Average wait time of connections before they are serviced by the connection pool.
`connectionrequestwaittime`	RangeStatistic	The longest and shortest wait times of connection requests. The current value indicates the wait time of the last request that was serviced by the pool.
`numconnfailedvalidation`	CountStatistic	Total number of connections in the connection pool that failed validation from the start time until the last sample time.
`numconnused`	RangeStatistic	Total number of connections that are currently being used, as well as information about the maximum number of connections that were used (the high watermark).
`numconnfree`	RangeStatistic	Total number of free connections in the pool as of the last sampling.
`numconntimedout`	CountStatistic	Total number of connections in the pool that timed out between the start time and the last sample time.
`numconncreated`	CountStatistic	Number of physical connections, in milliseconds, that were created since the last reset.
`numconndestroyed`	CountStatistic	Number of physical connections that were destroyed since the last reset.
`numconnacquired`	CountStatistic	Number of logical connections acquired from the pool.
`numconnreleased`	CountStatistic	Number of logical connections released to the pool.
`waitqueuelenght`	CountStatistic	Number of connection requests in the queue waiting to be serviced.

Connector Work Management Statistics (JMS)

Use the following dotted name pattern for JMS/Connector Service work management statistics:

```
server.connector-service.resource-adapter-1.work-management.statistic
```

JMS/Connector Service statistics available for connector work management are listed in the following table.

Table 8-13 Connector Work Management Monitoring Statistics (JMS)

Statistic	Data Type	Description
`activeworkcount`	RangeStatistic	Number of work objects executed by the connector.
`completedworkcount`	CountStatistic	Number of work objects that were completed.
`rejectedworkcount`	CountStatistic	Number of work objects rejected by the Payara Server.
`submittedworkcount`	CountStatistic	Number of work objects submitted by a connector module.
`waitqueuelength`	RangeStatistic	Number of work objects waiting in the queue before executing.
`workrequestwaittime`	RangeStatistic	Longest and shortest wait of a work object before it gets executed.

JVM Statistics

The JVM fits into the tree of objects as show in jvm-tree-hierarchy.

JVM Class Loading System Statistics

Use the following dotted name pattern for JVM class loading system statistics:

```
server.jvm.class-loading-system.statistic
```

With Java SE, additional monitoring information can be obtained from the JVM. Set the monitoring level to LOW to enable the display of this additional information. Set the monitoring level to HIGH to also view information pertaining to each live thread in the

system. More information about the additional monitoring features for Java SE is available in Monitoring and Management for the Java Platform .

The Java SE monitoring tools are discussed at `https://docs.oracle.com/en/java/javase/17/docs/specs/man/index.html`.

The statistics that are available for class loading in the JVM for Java SE are shown in the following table.

Table 8-14 JVM Monitoring Statistics for Java SE Class Loading

Statistic	Data Type	Description
`loadedclasscount`	CountStatistic	Number of classes that are currently loaded in the JVM
`totalloadedclasscount`	CountStatistic	Total number of classes that have been loaded since the JVM began execution
`unloadedclasscount`	CountStatistic	Number of classes that have been unloaded from the JVM since the JVM began execution

The statistics available for threads in the JVM in Java SE are shown in the following table.

Table 8-15 JVM Monitoring Statistics for Java SE - Threads

Statistic	Data Type	Description
`allthreadids`	StringStatistic	List of all live thread ids.
`currentthreadcputime`	CountStatistic	CPU time for the current thread (in nanoseconds) if CPU time measurement is enabled. If CPU time measurement is disabled, returns -1.
`daemonthreadcount`	CountStatistic	Current number of live daemon threads.
`monitordeadlockedthreads`	StringStatistic	List of thread ids that are monitor deadlocked.
`peakthreadcount`	CountStatistic	Peak live thread count since the JVM started or the peak was reset.

Statistic	Data Type	Description
`threadcount`	CountStatistic	Current number of live daemon and non-daemon threads.
`totalstartedthreadcount`	CountStatistic	Total number of threads created and/or started since the JVM started.

JVM Compilation System Statistics

Use the following dotted name pattern for JVM compilation system statistics:

```
server.jvm.compilation-system.statistic
```

The statistics that are available for compilation in the JVM for Java SE are shown in the following table.

Table 8-16 JVM Monitoring Statistics for Java SE Compilation

Statistic	Data Type	Description
`name-current`	StringStatistic	Name of the current compiler
`totalcompilationtime`	CountStatistic	Accumulated time (in milliseconds) spent in compilation

JVM Garbage Collectors Statistics

Use the following dotted name pattern for JVM garbage collectors statistics:

```
server.jvm.garbage-collectors.statistic
```

The statistics that are available for garbage collection in the JVM for Java SE are shown in the following table.

Table 8-17 JVM Monitoring Statistics for Java SE Garbage Collectors

Statistic	Data Type	Description
`collectioncount`	CountStatistic	Total number of collections that have occurred
`collectiontime`	CountStatistic	Accumulated time (in milliseconds) spent in collection

JVM Memory Statistics

Use the following dotted name pattern for JVM memory statistics:

```
server.jvm.memory.statistic
```

The statistics that are available for memory in the JVM for Java SE are shown in the following table.

Table 8-18 JVM Monitoring Statistics for Java SE Memory

Statistic	Data Type	Description
`committedheapsize`	CountStatistic	Amount of heap memory (in bytes) that is committed for the JVM to use
`committednonheapsize`	CountStatistic	Amount of non-heap memory (in bytes) that is committed for the JVM to use
`initheapsize`	CountStatistic	Size of the heap initially requested by the JVM
`initnonheapsize`	CountStatistic	Size of the non-heap area initially requested by the JVM
`maxheapsize`	CountStatistic	Maximum amount of heap memory (in bytes) that can be used for memory management
`maxnonheapsize`	CountStatistic	Maximum amount of non-heap memory (in bytes) that can be used for memory management
`objectpendingfinalizationcount`	CountStatistic	Approximate number of objects that are pending finalization
`usedheapsize`	CountStatistic	Size of the heap currently in use
`usednonheapsize`	CountStatistic	Size of the non-heap area currently in use

JVM Operating System Statistics

Use the following dotted name pattern for JVM operating system statistics:

```
server.jvm.operating-system.statistic
```

The statistics that are available for the operating system for the JVM machine in Java SE are shown in the following table.

Table 8-19 JVM Statistics for the Java SE Operating System

Statistic	Data Type	Description
`arch-current`	StringStatistic	Operating system architecture
`availableprocessors`	CountStatistic	Number of processors available to the JVM
`name-current`	StringStatistic	Operating system name
`version-current`	StringStatistic	Operating system version

JVM Runtime Statistics

Use the following dotted name pattern for JVM runtime statistics:

```
server.jvm.runtime.statistic
```

The statistics that are available for the runtime in the JVM runtime for Java SE are shown in the following table.

Table 8-20 JVM Monitoring Statistics for Java SE Runtime

Statistic	Data Type	Description
`classpath-current`	StringStatistic	Classpath that is used by the system class loader to search for class files
`inputarguments-current`	StringStatistic	Input arguments passed to the JVM; not including arguments to the `main` method
`managementspecversion-current`	StringStatistic	Management specification version implemented by the JVM
`name-current`	StringStatistic	Name representing the running JVM
`specname-current`	StringStatistic	JVM specification name
`specvendor-current`	StringStatistic	JVM specification vendor

Statistic	Data Type	Description
`specversion-current`	StringStatistic	JVM specification version
`uptime`	CountStatistic	Uptime of the JVM (in milliseconds)
`vmname-current`	StringStatistic	JVM implementation name
`vmvendor-current`	StringStatistic	JVM implementation vendor
`vmversion-current`	StringStatistic	JVM implementation version

Network Statistics

Network fits into the tree of objects as shown in network-tree-hierarchy.

Network Keep Alive Statistics

Use the following dotted name pattern for network keep alive statistics:

```
server.network.type-of-listener.keep-alive.statistic
```

Statistics available for network keep alive are shown in the following table.

Table 8-21 Network Keep Alive Statistics

Statistic	Data Type	Description
`countconnections`	CountStatistic	Number of connections in keep-alive mode.
`counttimeouts`	CountStatistic	Number of keep-alive connections that timed out.
`secondstimeouts`	CountStatistic	Keep-alive timeout value in seconds.
`maxrequests`	CountStatistic	Maximum number of requests allowed on a single keep-alive connection.
`countflushes`	CountStatistic	Number of keep-alive connections that were closed.
`counthits`	CountStatistic	Number of requests received by connections in keep-alive mode.
`countrefusals`	CountStatistic	Number of keep-alive connections that were rejected.

Network Connection Queue Statistics

Use the following dotted name pattern for network connection queue statistics:

```
server.network.type-of-listener.connection-queue.statistic
```

Statistics available for network connection queue are shown in the following table.

Table 8-22 Network Connection Queue Statistics

Statistic	Data Type	Description
`countopenconnections`	CountStatistic	The number of open/active connections
`countoverflows`	CountStatistic	Number of times the queue has been too full to accommodate a connection
`countqueued`	CountStatistic	Number of connections currently in the queue
`countqueued15minutesaverage`	CountStatistic	Average number of connections queued in the last 15 minutes
`countqueued1minuteaverage`	CountStatistic	Average number of connections queued in the last 1 minute
`countqueued5minutesaverage`	CountStatistic	Average number of connections queued in the last 5 minutes
`counttotalconnections`	CountStatistic	Total number of connectionsthat have been accepted
`counttotalqueued`	CountStatistic	Total number of connections that have been queued
`maxqueued`	CountStatistic	Maximum size of the connection queue
`peakqueued`	CountStatistic	Largest number of connections that were in the queue simultaneously
`tickstotalqueued`	CountStatistic	(Unsupported) Total number of ticks that connections have spent in the queue

Network File Cache Statistics

Use the following dotted name pattern for network file cache statistics:

```
server.network.type-of-listener.file-cache.statistic
```

Statistics available for network file cache are shown in the following table.

Table 8-23 Network File Cache Statistics

Statistic	Data Type	Description
`contenthits`	CountStatistic	Number of hits on cached file content
`contentmisses`	CountStatistic	Number of misses on cached file content
`heapsize`	CountStatistic	Current cache size in bytes
`hits`	CountStatistic	Number of cache lookup hits
`infohits`	CountStatistic	Number of hits on cached file info
`infomisses`	CountStatistic	Number of misses on cached file info
`mappedmemorysize`	CountStatistic	Size of mapped memory used for caching in bytes
`maxheapsize`	CountStatistic	Maximum heap space used for cache in bytes
`maxmappedmemorysize`	CountStatistic	Maximum memory map size used for caching in bytes
`misses`	CountStatistic	Number of cache lookup misses data type
`opencacheentries`	CountStatistic	Number of current open cache entries

Network Thread Pool Statistics

Use the following dotted name pattern for network thread pool statistics:

```
server.network.type-of-listener.thread-pool.statistic
```

Statistics available for network thread pool are shown in the following table.

Table 8-24 Network Thread Pool Statistics

Statistic	Data Type	Description
`corethreads`	CountStatistic	Core number of threads in the thread pool
`currentthreadcount`	CountStatistic	Provides the number of request processing threads currently in the listener thread pool

Statistic	Data Type	Description
`currentthreadsbusy`	CountStatistic	Provides the number of request processing threads currently in use in the listener thread pool serving requests
`maxthreads`	CountStatistic	Maximum number of threads allowed in the thread pool
`totalexecutedtasks`	CountStatistic	Provides the total number of tasks, which were executed by the thread pool

ORB Statistics (Connection Manager)

The ORB fits into the tree of objects as shown in orb-tree-hierarchy.

Use the following dotted name patterns for ORB statistics:

```
server.orb.transport.connectioncache.inbound.statistic
server.orb.transport.connectioncache.outbound.statistic
```

The statistics available for the connection manager in an ORB are listed in the following table.

Table 8-25 ORB Monitoring Statistics (Connection Manager)

Statistic	Data Type	Description
`connectionsidle`	CountStatistic	Total number of connections that are idle to the ORB
`connectionsinuse`	CountStatistic	Total number of connections in use to the ORB
`totalconnections`	BoundedRangeStatistic	Total number of connections to the ORB

Resource Statistics (Connection Pool)

By monitoring connection pool resources you can measure performance and capture resource usage at runtime. Connections are expensive and frequently cause performance bottlenecks in applications. It is important to monitor how a connection pool is releasing and creating new connections and how many threads are waiting to retrieve a connection from a particular pool.

The connection pool resources fit into the tree of objects as shown in resources-tree-hierarchy.

Use the following dotted name pattern for general connection pool statistics:

```
server.resources.pool-name.statistic
```

Use the following dotted name pattern for application-scoped connection pool statistics:

```
server.applications.application-name.resources.pool-name.statistic
```

Use the following dotted name pattern for module-scoped connection pool statistics:

```
server.applications.application-name.module-name.resources.pool-name.statistic
```

The connection pool statistics are shown in the following tables.

NOTE

In order to improve system performance, connection pools are initialized lazily; that is, a pool is not initialized until an application first uses the pool or the pool is explicitly pinged. Monitoring statistics for a connection pool are not available until the pool is initialized.

Table 8-26 General Resource Monitoring Statistics (Connection Pool)

Statistic	Data Type	Description
`averageconnwaittime`	CountStatistic	Average wait-time-duration per successful connection request
`connrequestwaittime`	RangeStatistic	Longest and shortest wait times, in milliseconds, of connection requests since the last sampling. current value indicates the wait time of the last request that was serviced by the pool
`numconnacquired`	CountStatistic	Number of logical connections acquired from the pool since the last sampling

Statistic	Data Type	Description
`numconncreated`	CountStatistic	Number of physical connections that were created by the pool since the last reset
`numconndestroyed`	CountStatistic	Number of physical connections that were destroyed since the last reset
`numconnfailedvalidation`	CountStatistic	Number of connections in the connection pool that failed validation from the start time until the last sampling time
`numconnfree`	RangeStatistic	Number of free connections in the pool as of the last sampling
`numconnnotsuccessfullymatched`	CountStatistic	Number of connections rejected during matching
`numconnreleased`	CountStatistic	Number of connections released back to the pool since the last sampling
`numconnsuccessfullymatched`	CountStatistic	Number of connections successfully matched
`numconntimedout`	CountStatistic	Number of connections in the pool that timed out between the start time and the last sampling time
`numconnused`	RangeStatistic	Number of connections that are currently being used, as well as information about the maximum number of connections that were used (high watermark)
`frequsedsqlqueries`	StringStatistic	List of the most frequently used SQL queries (Available only when SQL Tracing is enabled)
`numpotentialconnleak`	CountStatistic	Number of potential connection leaks
`numpotentialstatementleak`	CountStatistic	Number of potential statement leaks (Available only when Statement Leak Detection is enabled)

Statistic	Data Type	Description
`numstatementcachehit`	CountStatistic	Number of statements that were found in the statement cache (Available only when the Statement Cache is enabled)
`numstatementcachemiss`	CountStatistic	Number of statements that were not found in the statement cache (Available only when the Statement Cache is enabled)
`waitqueuelength`	CountStatistic	Number of connection requests in the queue waiting to be serviced

Table 8-27 Application Specific Resource Monitoring Statistics (Connection Pool)

Statistic	Data Type	Description
`numconnacquired`	CountStatistic	Number of logical connections acquired from the pool since the last sampling
`numconnreleased`	CountStatistic	Number of connections released back to the pool since the last sampling
`numconnused`	RangeStatistic	Number of connections that are currently being used, as well as information about the maximum number of connections that were used (high watermark)

Security Statistics

Security fits into the tree of objects as shown in security-tree-hierarchy.

EJB Security Statistics

Use the following dotted name pattern for EJB security statistics:

```
server.security.ejb.statistic
```

The statistics available for EJB security are listed in the following table.

Table 8-28 EJB Security Monitoring Statistics

Statistic	Data Type	Description
`policyconfigurationcount`	CountStatistic	Number of policy configuration
`securitymanagercount`	CountStatistic	Number of EJB security managers

Web Security Statistics

Use the following dotted name pattern for web security statistics:

```
server.security.web.statistic
```

The statistics available for web security are listed in the following table.

Table 8-29 Web Security Monitoring Statistics

Statistic	Data Type	Description
`websecuritymanagercount`	CountStatistic	Number of security managers
`webpolicyconfigurationcount`	CountStatistic	Number of policy configuration objects

Realm Security Statistics

Use the following dotted name pattern for realm security statistics:

```
server.security.realm.statistic
```

The statistics available for realm security are listed in the following table.

Table 3. Table 8-30 Realm Security Monitoring Statistics

Statistic	Data Type	Description
`realmcount`	CountStatistic	Number of realms

Thread Pool Statistics

The thread pool fits into the tree of objects as shown in thread-pool-tree-hierarchy.

Thread Pool Monitoring Statistics

Use the following dotted name pattern for thread pool statistics:

```
server.thread-pool.thread-pool.statistic
```

The statistics available for the thread pool are shown in the following table.

Table 8-31 Thread Pool Monitoring Statistics

Statistic	Data Type	Description
`averagetimeinqueue`	BoundedRangeStatistic	Average amount of time (in milliseconds) a request waited in the queue before being processed
`averageworkcompletiontime`	BoundedRangeStatistic	Average amount of 0time (in milliseconds) taken to complete an assignment
`currentbusythreads`	CountStatistic	Number of busy threads
`currentnumberofthreads`	BoundedRangeStatistic	Current number of request processing threads
`numberofavailablethreads`	CountStatistic	Number of available threads
`numberofworkitemsinqueue`	BoundedRangeStatistic	Current number of work items waiting in queue
`totalworkitemsadded`	CountStatistic	Total number of work items added to the work queue as of last sampling

JVM Statistics for Java SE-Thread Information

The statistics available for `ThreadInfo` in the JVM in Java SE are shown in the following table.

Table 8-32 JVM Monitoring Statistics for Java SE - Thread Info

Statistic	Data Type	Description
`blockedcount`	CountStatistic	Total number of times that the thread entered the `BLOCKED` state.
`blockedtime`	CountStatistic	Time elapsed (in milliseconds) since the thread entered the `BLOCKED` state. Returns -1 if thread contention monitoring is disabled.

Statistic	Data Type	Description
`lockname`	StringStatistic	String representation of the monitor lock that the thread is blocked to enter or waiting to be notified through the `Object.wait` method.
`lockownerid`	CountStatistic	ID of the thread that holds the monitor lock of an object on which this thread is blocking.
`lockownername`	StringStatistic	Name of the thread that holds the monitor lock of the object this thread is blocking on.
`stacktrace`	StringStatistic	Stack trace associated with this thread.
`threadid`	CountStatistic	ID of the thread.
`threadname`	StringStatistic	Name of the thread.
`threadstate`	StringStatistic	State of the thread.
`waitedtime`	CountStatistic	Elapsed time (in milliseconds) that the thread has been in a `WAITING` state. Returns -1 if thread contention monitoring is disabled.
`waitedcount`	CountStatistic	Total number of times the thread was in `WAITING` or `TIMED_WAITING` states.

Transaction Service Statistics

The transaction service allows the client to freeze the transaction subsystem in order to roll back transactions and determine which transactions are in process at the time of the freeze. The transaction service fits into the tree of objects as shown in transactions-service-tree-hierarchy.

Use the following dotted name pattern for transaction service statistics:

```
server.transaction-service.statistic
```

The statistics available for the transaction service are shown in the following table.

Table 8-33 Transaction Service Monitoring Statistics

Statistic	Data Type	Description
`activecount`	CountStatistic	Number of transactions currently active.
`activeids`	StringStatistic	The ID's of the transactions that are currently active. Every such transaction can be rolled back after freezing the transaction service.
`committedcount`	CountStatistic	Number of transactions that have been committed.
`rolledbackcount`	CountStatistic	Number of transactions that have been rolled back.
`state`	StringStatistic	Indicates whether the transaction has been frozen.

Web Statistics

The web module fits into the tree of objects as shown in web-tree-hierarchy.

Web Module Servlet Statistics

Use the following dotted name pattern for web module servlet statistics:

```
server.applications.web-module.virtual-server.servlet.statistic
server.applications.application.web-module.virtual-server.servlet.statistic
```

The available web module servlet statistics are shown in the following table.

Table 8-34 Web Module Servlet Statistics

Statistic	Data Type	Description
`errorcount`	CountStatistic	Cumulative number of cases where the response code is greater than or equal to 400.
`maxtime`	CountStatistic	Maximum amount of time the web container waits for requests.
`processingtime`	CountStatistic	Cumulative value of the amount of time required to process each request. The processing time is the average of request processing times divided by the request count.
`requestcount`	CountStatistic	The total number of requests processed so far.
`servicetime`	CountStatistic	Aggregate response time in milliseconds.

Web JSP Statistics

Use the following dotted name pattern for web JSP statistics:

```
server.applications.web-module.virtual-server.statistic
server.applications.application.web-module.virtual-server.statistic
```

The available web JSP statistics are shown in the following table.

Table 8-35 Web JSP Monitoring Statistics

Statistic	Data Type	Description
`jspcount-current`	RangeStatistic	Number of active JSP pages
`jsperrorcount`	CountStatistic	Total number of errors triggered by JSP page invocations
`jspreloadedcount`	CountStatistic	Total number of JSP pages that were reloaded
`totaljspcount`	CountStatistic	Total number of JSP pages ever loaded

Web Request Statistics

Use the following dotted name pattern for web request statistics:

```
server.applications.web-module.virtual-server.statistic
server.applications.application.web-module.virtual-server.statistic
```

The available web request statistics are shown in the following table.

Table 8-36 Web Request Monitoring Statistics

Statistic	Data Type	Description
`errorcount`	CountStatistic	Cumulative value of the error count, with error count representing the number of cases where the response code was greater than or equal to 400
`maxtime`	CountStatistic	Longest response time for a request; not a cumulative value, but the largest response time from among the response times
`processingtime`	CountStatistic	Average request processing time, in milliseconds
`requestcount`	CountStatistic	Cumulative number of the requests processed so far

Web Servlet Statistics

Use the following dotted name pattern for web servlet statistics:

```
server.applications.web-module.virtual-server.statistic
server.applications.application.web-module.virtual-server.statistic
```

The available web servlet statistics are shown in the following table.

Table 8-37 Web Servlet Monitoring Statistics

Statistic	Data Type	Description
`activeservletsloadedcount`	RangeStatistic	Number of currently loaded servlets
`servletprocessingtimes`	CountStatistic	Cumulative servlet processing times , in milliseconds
`totalservletsloadedcount`	CountStatistic	Cumulative number of servlets that have been loaded into the web module

Web Session Statistics

Use the following dotted name pattern for web session statistics:

```
server.applications.web-module.virtual-server.statistic
server.applications.application.web-module.virtual-server.statistic
```

The available web session statistics are shown in the following table.

Table 8-38 Web Session Monitoring Statistics

Statistic	Data Type	Description
`activatedsessionstotal`	CountStatistic	Total number of activated sessions
`activesessionscurrent`	RangeStatistic	Number of currently active sessions
`activesessionshigh`	CountStatistic	Maximum number of concurrently active sessions
`expiredsessionstotal`	CountStatistic	Total number of expired sessions
`passivatedsessionstotal`	CountStatistic	Total number of passivated sessions
`persistedsessionstotal`	CountStatistic	Total number of persisted sessions

Statistic	Data Type	Description
`rejectedsessionstotal`	CountStatistic	Total number of rejected sessions
`sessionstotal`	CountStatistic	Total number of sessions created

Configuring Jconsole to View Payara Server Monitoring Data

Java SE provides tools to connect to an MBean Server and view the MBeans registered with the server. JConsole is one such popular JMX Connector Client and is available as part of the standard Java SE distribution.When you configure JConsole for use with Payara Server, Payara Server becomes the JMX Connector's server end and JConsole becomes the JMX connector's client end.

To Connect Jconsole to Payara Server

Java SE 6 enhances management and monitoring of the virtual machine by including a Platform MBean Server and by including managed beans (MBeans) to configure the virtual machine.

To view all MBeans, Payara Server provides a configuration of the standard JMX connector server called System JMX Connector Server. As part of Payara Server startup, an instance of this JMX Connector Server is started. Any compliant JMX connector client can connect to the server using the JMX Connector Server.

By default, Payara Server is configured with a non-secure System JMX Connector Server. If this is an issue, the JMX connector can be removed. However, access can be restricted to a specific IP address (for example, the loopback address) by setting `address` to `locahost`.

- Start the domain. For instructions, see to-start-a-domain.
- Start JConsole using this format: JDK_HOME`/bin/jconsole`. For example:

  ```
  /usr/java/bin/jconsole
  ```

  The JConsole Connect to Agent window is displayed.

- Click the Remote tab and type the host name and port.
  Always connect remotely with JConsole, otherwise MBeans will not load automatically.
- Click Connect.
- In the Remote Process text box, specify the JMX Service URL.
  For example:

```
service:jmx:rmi:///jndi/rmi://localhost:8686/jmxrmi
```

The JMX Service URL is emitted by the server at startup, looking something like this:

```
[#|2009-12-03T10:25:17.737-0800|INFO|glassfishv3.0|
x..system.tools.admin.org.glassfish.server|_ThreadID=20;
_ThreadName=Thread-26;|JMXStartupService: Started JMXConnector, JMXService
URL = service:jmx:rmi://localhost:8686/jndi/rmi://localhost:8686/jmxrmi|#]
```

However, in most cases, simply entering `host:port` is fine, such as, 192.168.1.150:8686. The long Service URL is not needed.

NOTE Another host name can be substituted for `localhost`. The default port number (8686) could change if the `jmx-connector` configuration has been modified.

- Click Connect.
  In the JConsole window you will see all your MBeans, JVM information, and so on, in various tabs. Most of the useful MBeans are to be found in the `amx` and `java.lang` domains.

For more information about JConsole, see `http://docs.oracle.com/javase/8/docs/technotes/guides/management/jconsole.html`.

JMX Monitoring

Payara Server offers a customized JMX Monitoring Service. Once configured, Payara Server will monitor and log the values of all MBean attributes that have been added for

explicit monitoring.

The metrics are logged together in a single log message as a series of key-value pairs prefixed by the string `JMX-MONITORING:`.

NOTE	The default JMX listener is unauthenticated and open for local access. In an unsecure environment, open JMX ports should be disabled to prevent unwanted access.

To see a non-exhaustive inventory of existing MBeans that are currently available for Payara Server, see "here".

Enabling JMX Monitoring

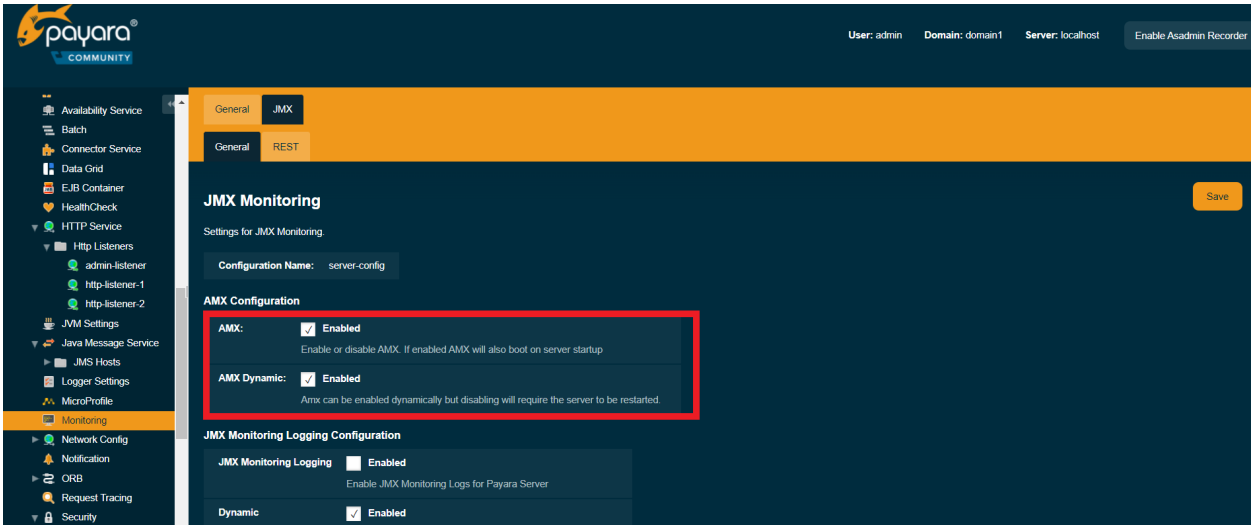
IMPORTANT	Payara Server uses the AMX API for working with *JMX MBeans*. AMX is not fully exposed by default and as such needs to be loaded to access most *JMX MBean* objects. The JMX Monitoring Service can be used without AMX but there is a limit to what can be monitored without it.

Boot AMX

To boot AMX at startup or independently of the JMX Monitoring service, run the command below:

```
asadmin set-amx-enabled --enabled true --dynamic true --target server
```

Boot AMX Using the Admin Console



Configure JMX Monitoring

You can enable JMX Monitoring using the `set-jmx-monitoring-configuration` asadmin command

JMX Monitoring can either be enabled for next startup or the service can be dynamically enabled on a running instance of Payara Server (provided a non-empty configuration exists at server startup).

To enable the service dynamically on the default server instance run the following command:

```
asadmin> set-jmx-monitoring-configuration --dynamic true --enabled true
```

Setting the JMX Monitoring Configuration

You can configure JMX Monitoring using the `set-jmx-monitoring-configuration` asadmin command.

Adding a Monitoring Attribute

To add the `HeapMemoryUsage` attribute to the list of MBean attributes to monitor using the service the following command can be used:

```
asadmin set-jmx-monitoring-configuration --addattribute
'attributeName=HeapMemoryUsage objectName=java.lang:type=Memory' --enabled false
```

Breaking this command down, two arguments have been used:

- `--addattribute`
- `--enabled`

Passing `--addattribute` to `set-jmx-monitoring-configuration` provides a way to add a new **MBean** attribute to monitor using the service. This argument takes in a string of space-delimited key-value pairs corresponding to the values listed earlier.

The `attributeName` and `objectName` fields are required, but `description` is not. Providing `attributeName=HeapMemoryUsage` denotes that the name of the **MBean** attribute to log is `HeapMemoryUsage`, while `objectName=java.lang:type=Memory` denotes the `ObjectName` of the **MBean** to look for the attribute on is `java.lang:type=Memory`.

The second argument, `--enabled`, is the only required option for the `asadmin` command. The only valid values to give this option are `true` or `false`. Passing `false` to the option will disable the logging service on next startup if it is currently enabled, but will otherwise do nothing.

Under this scenario the monitoring service has not been configured yet so `false` was passed.

Dealing with Composite Mbean Attributes

The **MBean** attribute added, `HeapMemoryUsage`, is a composite attribute.

It has metrics for the `commited`, `init`, `max` and `used` attributes. The monitoring service will by default monitor each metric and log it as `{$metric}{$attribute_name}:{$attribute_value}`.

If this is not the desired result, it is possible to monitor a single metric for a composite **MBean** attribute.

To monitor a single metric for the attribute the value of `attributeName` passed to the `--addattribute` option should be modified like so:

```
attributeName=HeapMemoryUsage.metric
```

So to log only the used heap memory the asadmin command would be:

```
asadmin set-jmx-monitoring-configuration --addattribute
'attributeName=HeapMemoryUsage.used objectName=java.lang:type=Memory' --enabled
false
```

Setting Logging Frequency

There are two configuration attributes related to the frequency at which log messages are written: `logfrequency` and `logfrequencyunit`. The first is a numerical value used for the rate, while the second is the unit for the rate. The default configuration is set to have a message logged every *15 seconds*.

If the value of `logfrequencyunit` is the default of `SECONDS` then to have the monitoring service log messages every one minute execute the following command:

```
asadmin set-jmx-monitoring-configuration --logfrequency 60 --enabled false
```

Rest Monitoring

This section covers how to use the REST monitoring functionality in Payara Server. The REST monitoring is implemented through an internal application that functions similarly to Jolokia, as it exposes MBeans over HTTP with a REST API to stop JMX monitoring requiring RMI.

The REST monitoring is hosted on the admin-listener (port `4848`) on the default context root, `/rest-monitoring`.

Configuring REST Monitoring

REST monitoring is configurable through several Asadmin CLI commands.

The `set-rest-monitoring-configuration` asadmin command is used to configure REST monitoring. This command requires the server to be running. Each configuration property can be configured individually or all together.

To enable REST monitoring, run the command like this:

```
asadmin set-rest-monitoring-configuration --enabled true
```

Getting REST Monitoring Configuration

```
asadmin> get-rest-monitoring-configuration
```

The `get-rest-monitoring-configuration` asadmin command is used to get the configuration of the REST monitoring. This command requires the server to be running.

If no target is specified, the command will get the REST monitoring configuration from the domain configuration (`server-config`). The `--target` option retrieves the REST monitoring from another instance or cluster.

Here's an example of how to use the command:

```
asadmin> get-rest-monitoring-configuration --target server-config

Enabled    Rest Monitoring Application Name    Context Root       Security Enabled
true       __restmonitoring                    /rest-monitoring   false
```

Rest Monitoring Security

IMPORTANT	To enable security for the REST Monitoring application, you'll have to enable secure administration first.

```
asadmin set-rest-monitoring-configuration --securityEnabled=true
```

The `securityEnabled` property defines whether the REST monitoring application is available over HTTPS. Since the REST monitoring application is hosted on the `admin-listener`.

Enabling secure admin can be done by first configuring the admin password with the `change-admin-password` command, then using the `enable-secure-admin` command.

NOTE Remember that enabling secure admin requires a server restart.

When `securityenabled` is set to true, a username and password is required to use REST monitoring.

NOTE The default username is set to `payara` and the default password is set to `rest`. The user `payara` is defined in the `file` realm.

The password can be changed with the following command:

```
asadmin update-file-user --groups=rest --target=server-config --authrealmname=file
payara
```

This updates the `payara` user, which is part of the `rest` group. Any users who are able to log in to the REST monitoring must also be defined in the `file` realm and be part of the `rest` group.

New users can be created by using the following command:

```
asadmin create-file-user --groups=rest --target=server-config --authRealmName=file
<username>
```

Performing READ Operations

REST Monitoring supports a subset of operations in the Jolokia API.

The `read` operation reads the details of the requested MBean. The `read` operation accepts GET requests on URLs in the following format:

```
<REST_API_URL>/read/${mbean-name}/${attribute-name}
```

A list of attribute names can be found in the request of an empty attribute name under `value`.

Example Execution

To read the MBean `java.lang:type=Memory` using the default configuration, you

would make a GET request to: `http://localhost:4848/rest-monitoring/rest/read/java.lang:type=Memory`.

Example Output

```
{
  "request": {
    "mbean": "java.lang:type=Memory",
    "type": "read"
  },
  "value": {
    "HeapMemoryUsage": {
      "committed": 450363392,
      "init": 264241152,
      "max": 477626368,
      "used": 97480984
    },
    "ObjectPendingFinalizationCount": 0,
    "NonHeapMemoryUsage": {
      "committed": 139460608,
      "init": 2555904,
      "max": -1,
      "used": 122389432
    },
    "Verbose": false,
    "ObjectName": "java.lang:type=Memory"
  },
  "timestamp": 1502799650273,
  "status": 200
}
```

Performing Bulk READ Operations

It is possible to execute bulk operations using the REST monitoring API. To do this, issue a POST request to the REST API URL with the following JSON structure as the body payload.

The payload is a JSON array consisting of objects of type, MBean, and attribute JSON objects. You may send a single operation request instead of an array also.

```
[
  {
    "type" : "<OPERATION_TYPE>",
    "mbean" : "<MBEAN_NAME>",
    "attribute" : "<ATTRIBUTE_NAME>"
  },
  {
    "type" : "<OPERATION_TYPE>",
    "mbean" : "<MBEAN_NAME>",
    "attribute" : "<ATTRIBUTE_NAME>"
```

```
  }
]
```

- **type** - The 'type' of operation to execute, e.g. read, search, write, etc.
- **mbean** - The MBean attribute which the operation will be executed on.
- **attribute** - The MBean attribute upon which the operation will be executed. If omitted, all attributes of the MBean will be involved in the operation.

Example Execution

Using `curl` and a sample REST API URL of `http://localhost:4848/rest-monitoring/rest`

```
curl -X POST http://localhost:4848/rest-monitoring/rest/ \
  -H 'Content-Type: application/json' \
  -d '[
    {
        "mbean": "java.lang:type=Compilation",
        "type": "read"
    },{
        "mbean": "java.lang:type=Runtime",
        "attribute" : "Uptime",
        "type": "read"
    }
]'
```

Example Output

```
[
    {
        "request": {
            "mbean": "java.lang:type=Compilation",
            "type": "read"
        },
        "value": {
            "Name": "HotSpot 64-Bit Tiered Compilers",
            "CompilationTimeMonitoringSupported": true,
            "TotalCompilationTime": 106363,
            "ObjectName": "java.lang:type=Compilation"
        },
        "timestamp": 1529353755633,
        "status": 200
    },
    {
        "request": {
            "mbean": "java.lang:type=Runtime",
            "attribute": "Uptime",
            "type": "read"
        },
```

```
        "value": 8541422,
        "timestamp": 1529353755636,
        "status": 200
    }
]
```

Administering the Healthcheck Service

Payara Server includes a HealthCheck Service that is disabled by default. When enabled it can periodically check the following performance metrics:

- Host CPU Usage
- Host Memory Usage
- Payara Server's JVM Garbage Collections
- Payara Server's JVM Heap Usage
- CPU Usage of individual threads
- Detects stuck threads
- Monitors metrics exposed by MicroProfile Metrics

If there is a problem with any of these metrics, and they exceed a configurable threshold then a `GOOD`, `WARNING` or `CRITICAL` event notification is sent to the Notification Service.

This allows operations teams to rapidly detect problems or work out what happened after these problems have occurred.

NOTE	It is possible to display metrics produced by HealthCheck service on MicroProfile Health rest-endpoints.
IMPORTANT	The Host Memory Usage currently only works on Linux and BSD operating system derivatives.

If the log-notifier is enabled, such events will be presented in the server's log file like in the following sample:

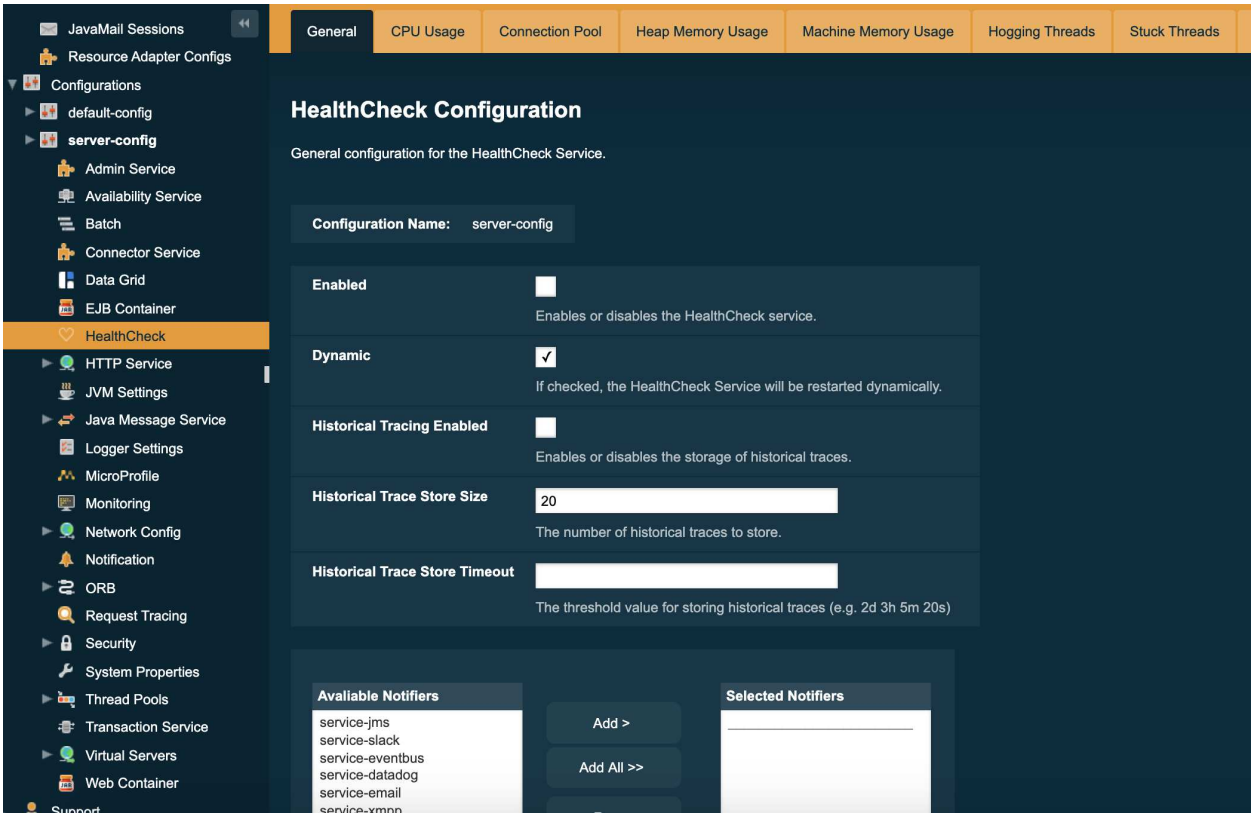
```
[2016-05-24T03:52:28.690+0000] [Payara 7.2.0] [INFO]
[fish.payara.nucleus.healthcheck.HealthCheckService] [tid: _ThreadID=72
_ThreadName=healthcheck-service-3 [timeMillis: 1464061948690] [levelValue: 800] [[
CPUC:Health Check Result:[[status=WARNING, message='CPU%: 75.6, Time CPU used: 267
milliseconds'']']]]  [2016-05-24T21:11:36.579+0000] [Payara 4.1] [SEVERE]
[fish.payara.nucleus.healthcheck.HealthCheckService] [tid: _ThreadID=71
_ThreadName=healthcheck-service-3] [timeMillis: 1464124296579] [levelValue: 1000]
[[ HOGT:Health Check Result:[[status=CRITICAL, message='Thread with <id-name>: 145-
testing-thread-1 is a hogging thread for the last 59 seconds 999
milliseconds'']']]]
```

Healthcheck Service Configuration

As with any other services available in Payara Server, the HealthCheck service can be configured by using the web console or by using administration commands.

Using the Web Administration Console

To configure the HealthCheck Service in the Administration Console, go to *Configuration* → *[instance-configuration (like server-config)]* → *Healthcheck*:



Check the **Enabled** box (and the **Dynamic** box too if you don't want to restart the domain) to switch the HealthCheck service on.

The general settings of the service are:

- **Threshold Unit**: This defines the time duration per unit. The accepted options are any valid java.util.concurrent.TimeUnit values.

NOTE	The default value is *SECONDS*.

- **Threshold Value**: This defines the number of units beyond which a request will be traced.
- **Store Historic Traces**: When present, enables the storage of the slowest historical request trace events.
- **Historic Trace Store Size**: Determines the number of historical trace events that can be stored in memory when historical storing is enabled. The default value is **20** records

Aside from this configuration settings, you can also define which notifiers will be used to relay the HealthCheck events by moving them to the **Active Notifiers** box.

IMPORTANT	Keep in mind that for HealthCheck events to be relayed to the active notifiers, both the "Notification Service" and each respective notifier must be enabled first.

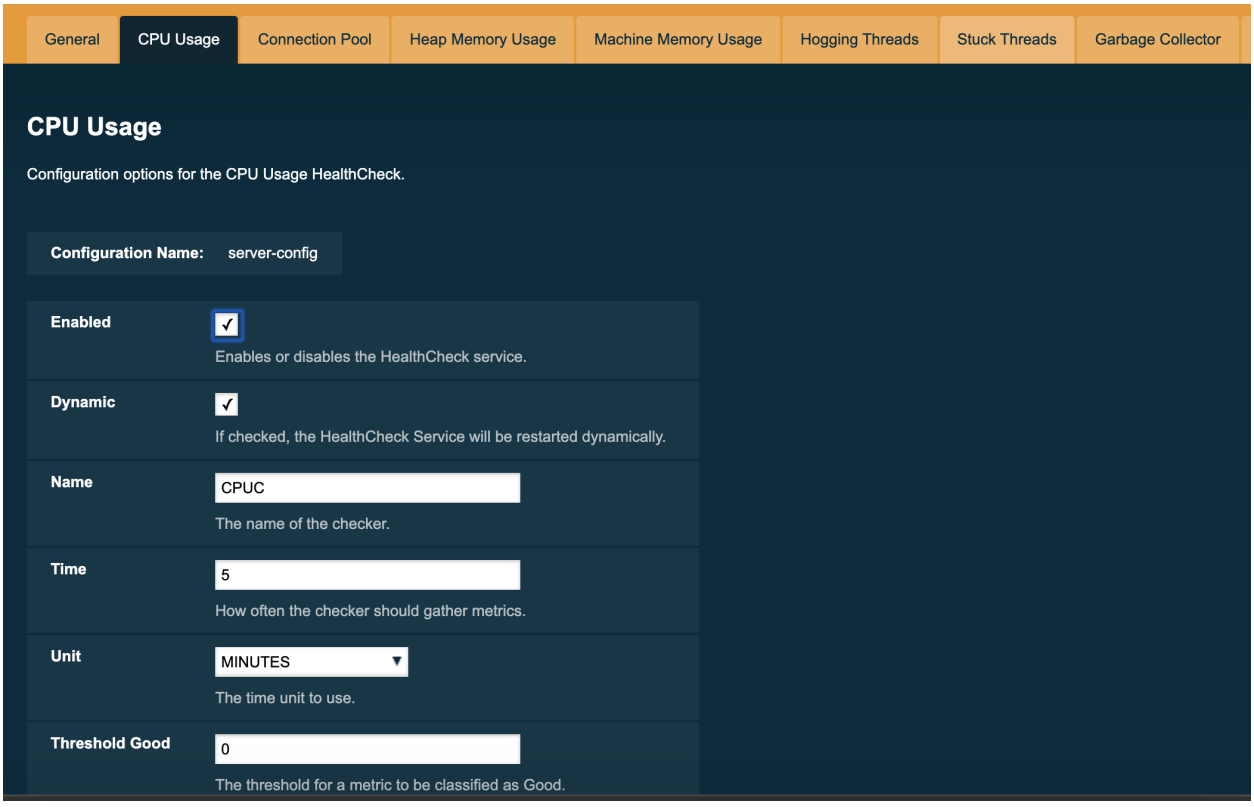
TIP	You don't need to manually add each notifier on this screen. When enabling a notifier on its configuration screen, the server will **automatically** add it to the list of active notifiers for the HealthCheck service. This same result occurs when enabling the notifier using the appropriate `asadmin` command.

Configuring the Available Checkers

Each of the available checkers that are used to determine the server's health can also be configured separately from each other on the admin console. The list of the available checkers is as follows:

CPU Usage: Calculates the CPU usage and prints out the percentage along with the usage time. **Connection Pool**: Calculates the ratio of free/used connections available for all JDBC connections pool and prints the percentage of used connections for each active pool. **Heap Memory Usage**: Calculates the heap memory usage and prints out the percentage along with initial and committed heap sizes. **Machine Memory Usage**: Calculates the machine memory usage and prints out the percentage along with the total and used physical memory size. **Hogging Threads**: Identifies the threads that are hogging the CPU. **Stuck Threads**: Identifies the threads that are stuck for a specified period of time. **Garbage Collector**: Calculates and prints out how many times GC is executed with its elapsed time. The percentage is the ratio of time spent in GC and total time passed.

You can configure the settings for each checker on the respective tab in the web console. Here's a sample image with the current configuration for the **CPU Usage** checker:



Common Healthcheck Service Checkers Configuration

The following are the configurable attributes available to **ALL** the HealthCheck Service performance metric checkers:

Enabled: Determines whether the checker is enabled. ***Dynamic***: Determine whether the changes done to the checker's configuration are applied immediately or after the server/instance's restart. ***Name***: The name or label that the checker will use to identify itself on the notification events. The default names for all checkers are the following:

+

Checker	Default name
CPU Usage	`CPUC`
Connection Pool	`CONP`
Heap Memory Usage	`HEAP`
Machine Memory Usage	`MEMM`
Hogging Threads	`HOGT`
Stuck Threads	`STUCK`
Garbage Collector	`GBGC`

Time: The time interval value (as an `Integer`) specified in given unit to execute the checker for the metric. The default value is **5**. ***Unit***: This defines the time duration per unit. The accepted options are any valid java.util.concurrent.TimeUnit values.

+ The default value is **MINUTES**.

Threshold Range Configuration for Healthcheck Service Checkers

The following attributes are available to the **CPU Usage**, **Connection Pool**, **Heap Memory Usage**, **Machine Memory Usage** and **Garbage Collector** checkers:

Threshold Good: The upper numeric boundary (valid `Integer`) of the metric used by the checker for the notification event to be classified as **GOOD**. It has **0** as default value. ***Threshold Warning***: The upper numeric boundary (valid `Integer`) of the metric used by

the checker for the notification event to be classified as **WARNING**. It has **50** as default value. ***Threshold Critical***: The upper numeric boundary (valid `Integer`) of the metric used by the checker for the notification event to be classified as **CRITICAL**. It has **80** as default value.

The threshold values range (GOOD - WARNING - CRITICAL) are used to correctly warn users of the health of a specific metric depending on their value when they are measured based on the checking frequency.

For example, if the **CPU Usage** checker is configured with the default threshold values, and at measuring time, the CPU is performing at `76,8%`. Then this notification event would be generated:

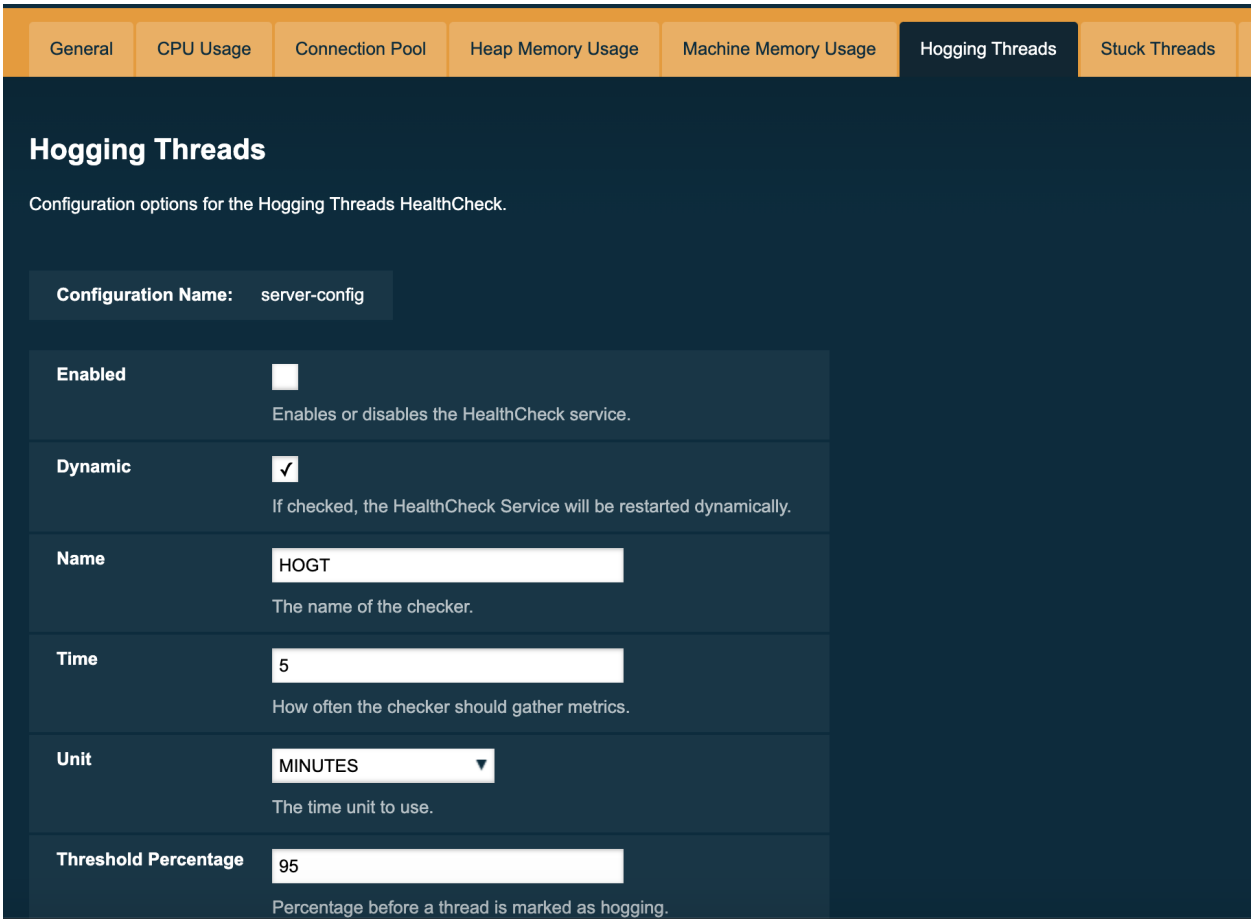
```
Health Check notification with severity level: WARNING – CPUC:Health Check
Result:[[status=WARNING, message='CPU%: 76.8, Time CPU used: 171 milliseconds'']']
```

Special Healthcheck Service Checkers Configuration

The **Hogging Threads** and the **Stuck Threads** checkers are special on their configuration.

They do not have a threshold range configuration, instead opting for different attributes.

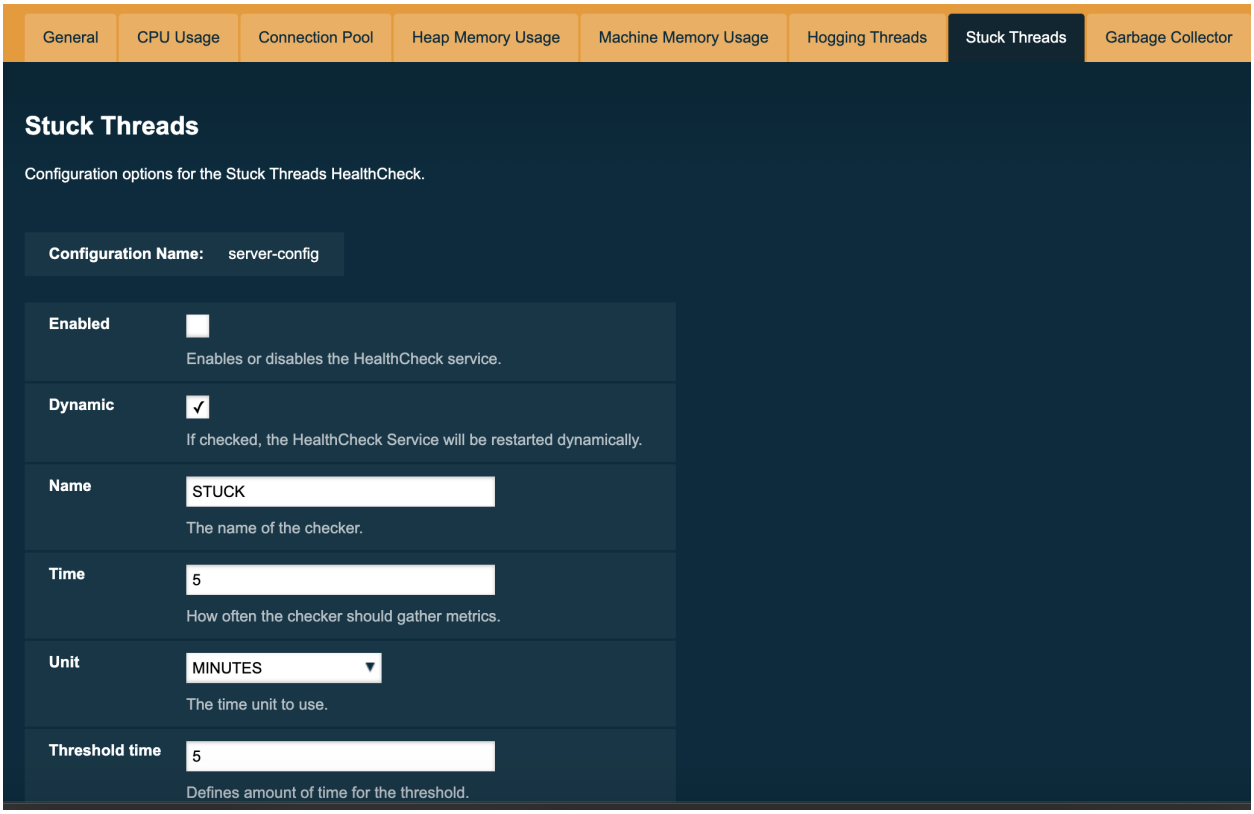
Here's a configuration sample of the **Hogging Threads** checker:



The following are the attributes used to configure this checker:

Threshold Percentage: Defines the minimum percentage needed to decide if the thread is hogged CPU-wise. The percentage is calculated with the ratio of elapsed CPU time to checker execution interval. Its default value is **95**. ***Retry Count***: Represents the count value that should be reached by the hogged thread in order for the service to send notifications. Its default value is **3**

And here's a configuration sample for the **Stuck Threads** checker:



The following are the attributes used to configure this checker:

Threshold Time: Defines the time value for which a thread can be non-responsive before it is considered stuck. It's default value is **5**. ***Threshold Unit***: Defines the time unit for the value of the **Threshold Time** field. It's default value is **Minutes**.

From the Command Line

You can configure the HealthCheck Service by using the corresponding Asadmin CLI commands.

To enable and configure the general settings of the Healthcheck service, use the `set-healthcheck-configuration` command. For example, to enable the service such that it will only activate from the next time the server is restarted and to configure the historic trace settings and notifiers used by it run the following command:

```
asadmin set-healthcheck-configuration --enabled=true --dynamic=true --historic
-trace-enabled=true --historic-trace-store-size=20 --set-notifiers=log-
notifier,jms-notifiers
```

To configure specific settings of the various performance metric service checkers, use the `set-healthcheck-service-configuration`. This command has multiple settings to configure a specific checker's sampling frequency, threshold values (where applicable) and other specific criteria to each checker service.

As an example, the following command will enable the Garbage Collector checker service and activate it without needing a server restart would be as follows:

```
asadmin set-healthcheck-service-configuration --enabled=true --service=gc --
dynamic=true
```

A more complex example would be the following command that configures the stuck threads checker to check every 30 seconds for any threads which have been stuck for more than 5 minutes:

```
asadmin set-healthcheck-service-configuration --service=st --enabled=true --
dynamic=true --time=30 --time-unit=SECONDS --stuck-threads-threshold=5 --stuck
-threads-threshold-unit=MINUTES
```

For more information and examples for how to use these Asadmin CLI commands, see the "`set-healthcheck-configuration`" and "`set-healthcheck-service-configuration`" entries in the Command Reference section.

Administering the Request Tracing Service

The Request Tracing Service provides tracing facilities for multiple protocols and process communications done by the components of deployed applications.

The service helps users to detect application slowness and performance degradation by logging requests that exceed a given threshold. The trace data from long-running requests gives insight to solving bottlenecks and other performance issues.

The following types of requests are traced if the service is enabled:

- JAX-RS endpoints (*)
- JAX-RS client calls (*)

- MicroProfile REST Client calls (*)
- Servlets (HTTP requests)
- SOAP Web Services (*)
- WebSockets
- EJB timers execution
- Inbound JMS messages received by an MDB
- JBatch jobs creation
- Tasks executed by injected Managed Executors.
- Remote Invocations of EJBs (*)

NOTE	Traces of request types marked in (*) are using higher-level APIs and are also available as OpenTelemetry Spans.

Traces have form of nested spans following the model found in modern tracing APIs such as **OpenTelemetry** and **OpenTracing**.

The OpenTracing specification, found here, gives a more detailed description of what Spans are and the general trace format. Alternatively you can view a short summary terminology

The Request Tracing service has the concept of sample rates, affording you extra options to limit the amount of traces processed. Azul Payara offers two options:

- **Flat probability sampling**, which simply applies a percentage chance that a trace will not be traced.
- An **adaptive solution**, which allows you to configure a target number of traces to sample over a given time period (e.g. sample 6 traces every 10 minutes).

An option to perform this sampling before or after the threshold checks is offered, allowing you to either determine if a trace should be sampled at its inception, or only after determining that a trace has breached your configured thresholds respectively.

Opentelemetry Support

As Azul Payara offers support for MicroProfile Telemetry, it is possible to manually instrument your application code for creating spans.

For the full details, see the opentelemetry-and-opentracing in the Azul Payara Application Development section.

Opentracing Support

In-line with Azul Payara's support for "MicroProfile OpenTracing", there's added support for OpenTracing.io code instrumentation.

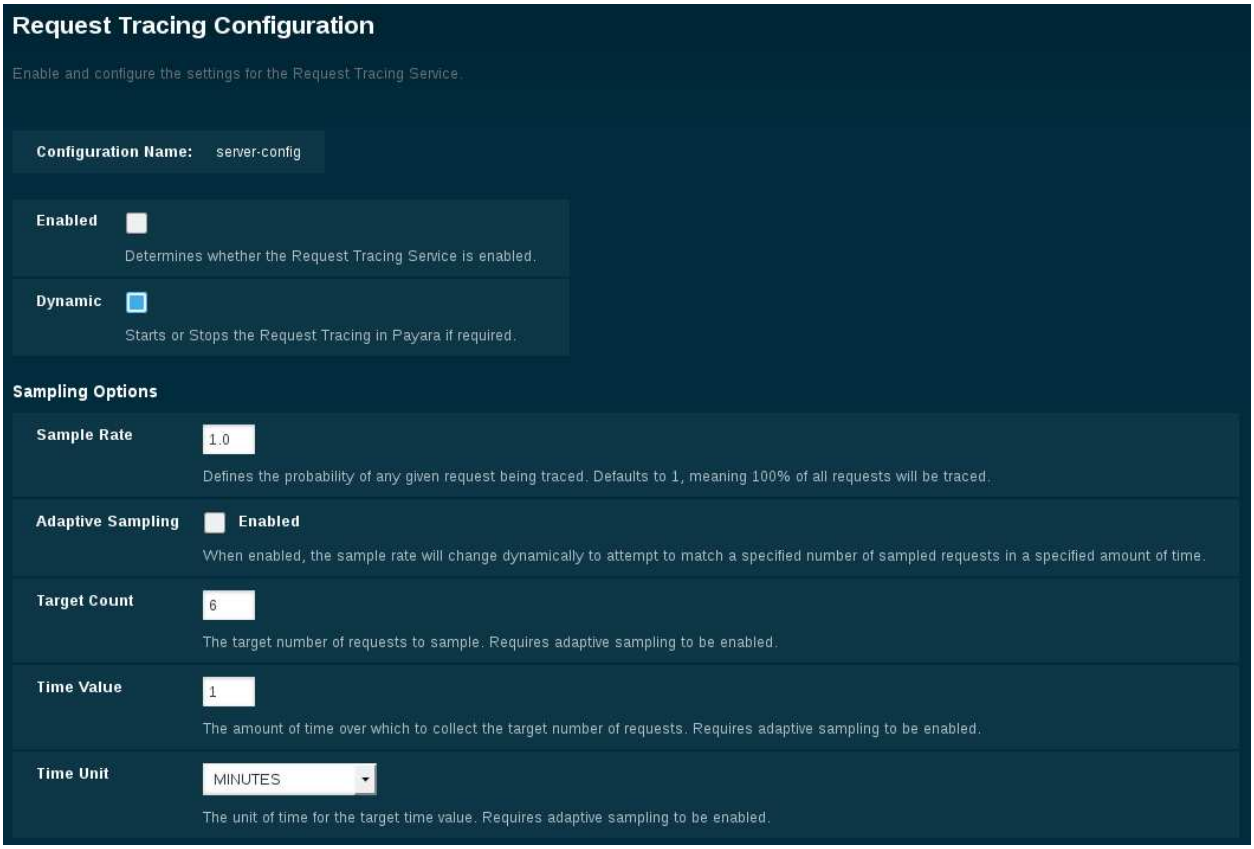
NOTE	Since OpenTracing is just a wrapper around OpenTelemetry, it is advised to use OpenTelemetry APIs instead.

Configuration

As with any other services available in Payara Server, the Request Tracing service can be configured by using the Admin Console or via the Asadmin CLI.

Using the Web Administration Console

To configure the Request Tracing Service in the Administration Console, go to *Configuration* → *[instance-configuration (like server-config)]* → *Request Tracing*:



Check the **Enabled** box (and the **Dynamic** box too if you don't want to restart the domain) to switch the request tracing service on.

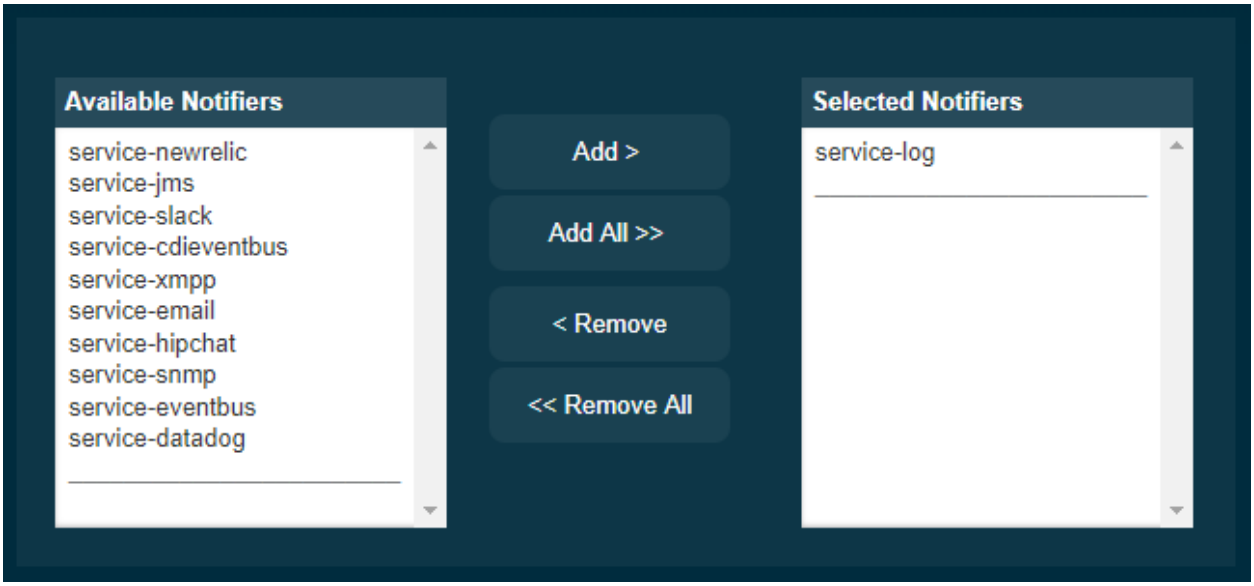
You can configure the request tracing service settings using the following attributes:

- **Sample Rate**: This defines the percentage chance of a trace being sampled (as in, not discarded). This accepts a value between 0 and 1, with a value of 1 meaning 100% of traces are sampled.
- **Adaptive Sampling**: This option is an alternative to using Sample Rate, allowing the sample rate to change in accordance with how many traces are being sampled over a period of time.
- **Target Count**: This defines the number of sampled traces that the adaptive sampling shall attempt to hit over a period of time.
- **Time Value**: This defines the period of time that the adaptive sampling will try to hit the *Target Count* in.

- **Time Unit**: This defines the time unit of the *Time Value* attribute.
- **Applications Only**: When enabled, traces that originate from outside deployed applications (e.g. the Admin Console) will not be sampled.
- **Threshold Value**: This defines the number of units beyond which a request will be traced.
- **Threshold Unit**: This defines the time duration per unit. The accepted options are any valid java.util.concurrent.TimeUnit values. The default value is *SECONDS*.
- **Sample Rate First**: When enabled, the sample rate will be checked as a trace begins, meaning that traces that would have broken the thresholds are potentially discarded. When disabled, the sample rate is checked after a trace has completed, meaning that you're only sampling traces that have breached your threshold values. Having this disabled will potentially give more accurate trace data, at the expense of a small extra performance overhead.
- **Reservoir Sampling**: When disabled, the trace stores will contain the longest request traces. When enabled, the chance of any sampled request being in the trace stores becomes constant. Note that this applies to both trace stores in most circumstances it doesn't make sense to have this enabled if *Store Historic Traces* is also enabled.
- **Trace Store Size**: The maximum number of traces to be stored in the local rolling trace store. A store shared in a cluster uses the maximum setting of this property of any instance in the cluster with request tracing enabled.
- **Trace Store Timeout**: The timeout value for when traces will be removed from the rolling trace store. This is ignored when reservoir sampling is enabled.
- **Store Historic Traces**: When enabled, a second trace store is created to store the longest historical request traces.
- **Historic Trace Store Size**: Determines the number of historical trace events that can be stored in memory when historical storing is enabled. The default value is **20** records. A store shared in a cluster uses the maximum setting of this property of any instance in the cluster with request tracing enabled.

- **Historic Trace Store Timeout**: Determines how long before Request Tracing messages are removed. This field is specified in the format "%d %h %m %s" where % is an integer. Any of the units can also be omitted.

Aside from this configuration settings, you can also define which notifiers will be used to relay the request tracing events by moving them to the **Selected Notifiers** box.



IMPORTANT	Keep in mind that for request tracing events to be relayed to the selected notifiers, both the "Notification Service" and each respective notifier must be enabled first.

TIP	You don't need to manually add each notifier on this screen. When enabling a notifier on its configuration screen, the server will **automatically** add it to the list of selected notifiers for the Request Tracing service. This same result occurs when enabling the notifier using the appropriate *asadmin* command.

From the Command Line

You can configure the Request Tracing Service by using these administrations commands:

- "`get-requesttracing-configuration`"
- "`set-requesttracing-configuration`"

Administering the Notification Service

Payara Server comes with a general Notification service that can be used to log events which come from other services, such as the "Monitoring Service", the "HealthCheck service" or the "Request Tracing Service".

Notification Service Configuration Commands

The following Asadmin CLI subcommands can be used to configure the notification service in general:

- "`notification-configure`"
- "`list-notifiers`"
- "`get-notification-configuration`"
- "`set-notification-configuration`"

Notifiers

The Notification service provides the ability to disseminate notification events through various channels or **notifiers**, which relay them to different destinations.

Filtering

All notifiers make use of a filtering system to accept or ignore notification events. Events are emitted with a severity level of `INFO`, `WARNING`, or `SEVERE`. Notifiers may be individually configured to filter out events which are below a certain severity level.

NOTE	A notifier with a filter level of `WARNING` (the default) will disregard events at the `INFO` level, and accept event at the `WARNING` and `SEVERE` levels.

The following is a list of the default set of notifiers available in Payara Server.

Log Notifier

The log notifier is the only notifier that is enabled by default when the notification

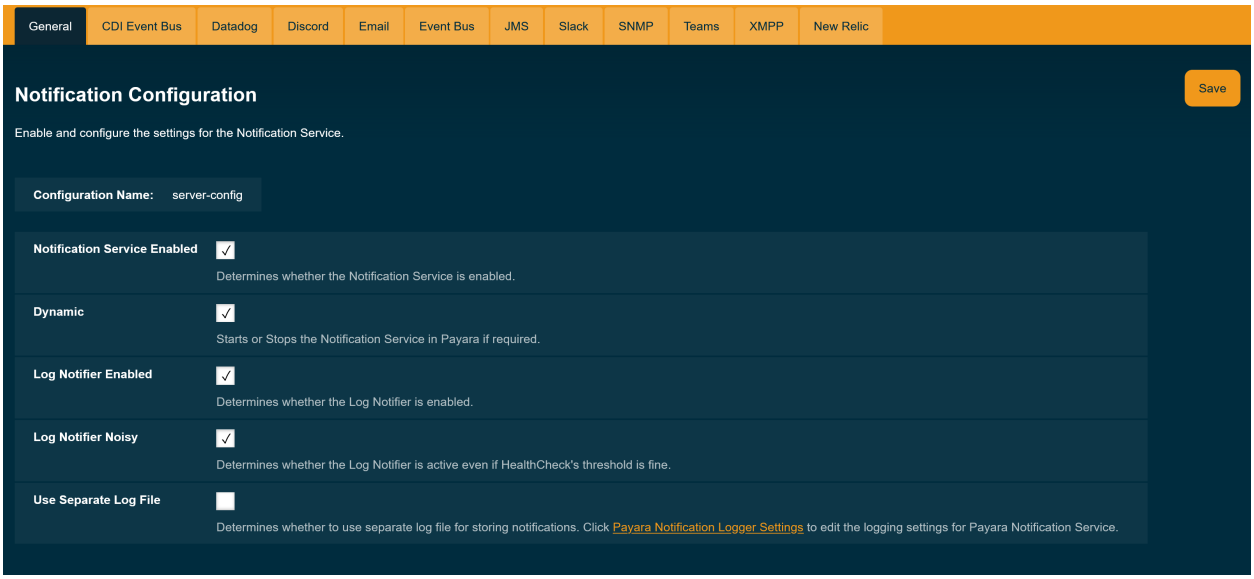
service is activated.

The default output for the log notifier is the configured instance's log file, which is either the *server.log* file or *cluster.log* file, depending on the instance configuration.

> **NOTE** The log notifier only handles the output of notifications from the notification service sent to the log and is not responsible for any other log output.

Configuration

Because the log notifier is the default notifier for the notification service it can be configured on the same page as the notification service, as shown in the image below:



For any new domain, the notification service will be disabled by default. It can be enabled through the admin console as shown above, or via the Asadmin CLI.

To configure the log notifier through the Asadmin CLI, use the `set-log-notifier-configuration` subcommand:

```
asadmin> set-log-notifier-configuration --enabled=true --dynamic=true
```

To get the current configuration of the log notifier, use the `get-log-notifier-configuration` subcommand:

```
asadmin> get-log-notifier-configuration

Enabled  Filter   Use Separate Log File
true     WARNING  false
Command get-log-notifier-configuration executed successfully.
```

See notification-logging for more information on how to configure specific details of the server logging settings.

JMS Notifier

The JMS notifier makes use of a JMS queue to send notification events via standard Java messages. These messages can then be consumed by a Message Driven Bean or any other JMS compatible client implementation.

Requirements

To use JMS as a notification target, you will need the following a JMS-compatible message broker.

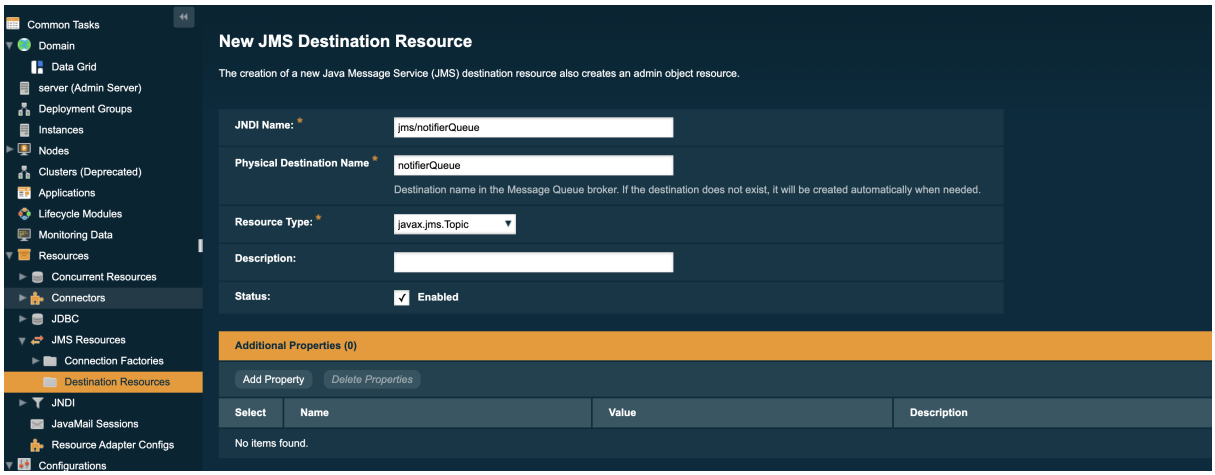
IMPORTANT	The Payara Server Web Profile distribution does not come with JMS capabilities, so this notifier isn't supported in this profile.

NOTE	The easiest way to configure the JMS notifier is to use the embedded OpenMQ broker, since Payara Server already comes with it by default.

JMS Notifier Configuration

The notifier configuration requires following a few steps:

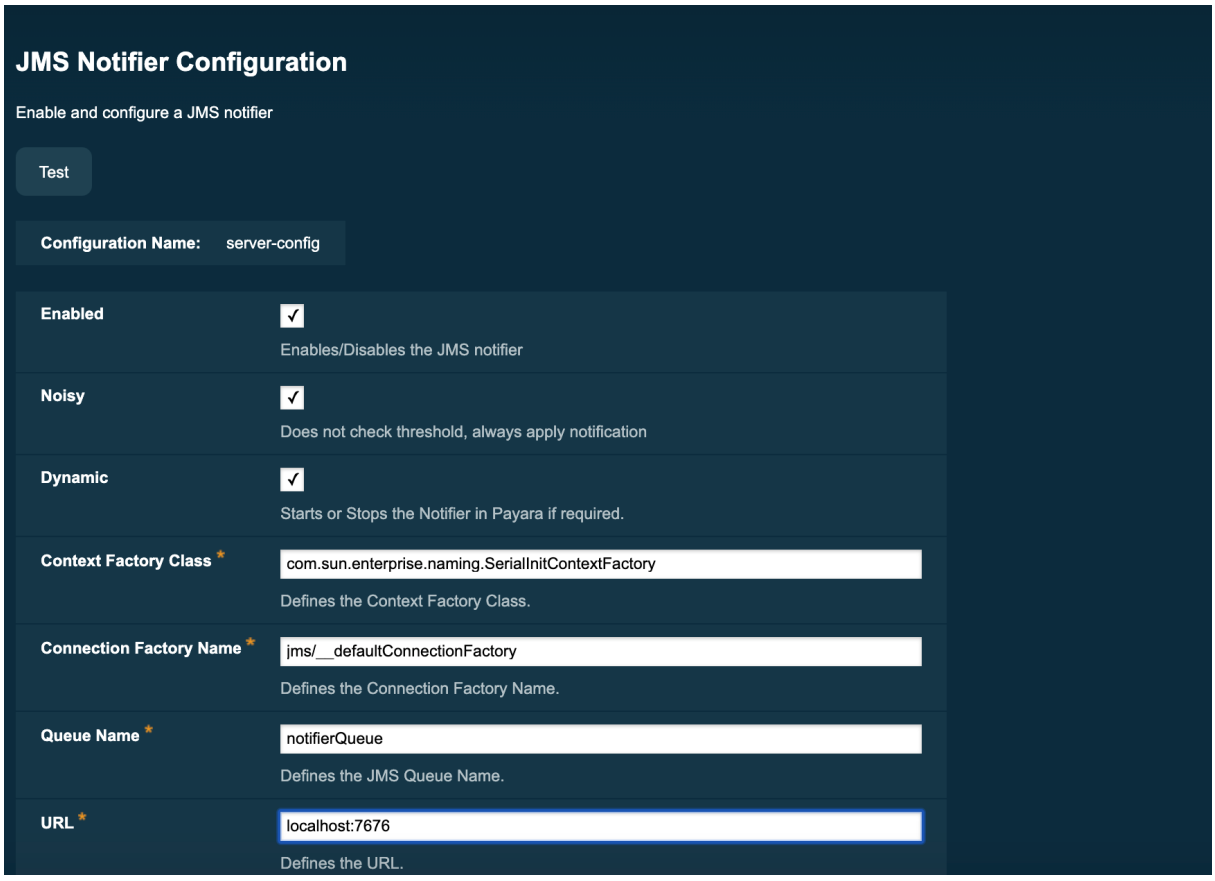
1. Create a new **JMS Queue** to receive the notification messages from the notifier:



To make this change via the Asadmin CLI, use the following subcommand which mirrors the above screenshot:

```
asadmin create-jms-resource --enabled=true --property=Name=notifierQueue
--restype=jakarta.jms.queue jms/notifierQueue
```

2. Set the JMS Notifier settings:



The above example uses the embedded OpenMQ broker's default configuration.

To make this change via the Asadmin CLI, use the following command which mirrors the above screenshot:

```
asadmin set-jms-notifier-configuration --dynamic=true --enabled=true
--contextFactoryClass=com.sun.enterprise.naming.SerialInitContextFactory
--target=server-config --queueName=notifierQueue --url=localhost:7676
--connectionFactoryName=jms/_defaultConnectionFactory
```

Verifying the Configuration

For verification purposes, it may be useful to enable a service to push notifications through the JMS notifier to demonstrate that it is working. To do this, you will also need to set up a Message-Driven Bean (MDB) to consume the notifications and demonstrate that they are being stored in the queue.

1. To consume the messages using a Jakarta Messaging compliant MDB, the following class will work for a pre-configured queue named `notifierQueue`:

```
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue
= "jms/notifierQueue"),
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"jakarta.jms.Queue"),
    @ActivationConfigProperty(propertyName = "destination", propertyValue =
"notifierQueue"),
})
public class NotificationConsumer implements MessageListener {

    @Override
    public void onMessage(Message message) {
        try {
            System.out.println("Message received: " + message.getBody(String
.class));
        } catch (JMSException ex) {}
    }
}
```

2. View the result of the MessageDrivenBean's `onMessage()` command. In this example, the CPU metric of the health check service was configured to notify every *5 seconds*, so the result of simply printing to `System.out` is following log entries:

```
[2017-02-24T14:25:02.019+0000] [INFO] []
```

```
[fish.payara.nucleus.healthcheck.HealthCheckService] [tid: _ThreadID=151
_ThreadName=admin-thread-pool::admin-listener(9)] [timeMillis: 1487946302019]
[levelValue: 800] [[Scheduling Health Check for task: CPUC]]

[2017-02-24T14:25:02.019+0000] [INFO] []
[fish.payara.nucleus.healthcheck.HealthCheckService] [tid: _ThreadID=151
_ThreadName=admin-thread-pool::admin-listener(9)] [timeMillis: 1487946302019]
[levelValue: 800] [[Payara Health Check Service Started.]]

[2017-02-24T14:25:02.376+0000] [INFO] [] [] [tid: _ThreadID=48 _ThreadName=p:
thread-pool-1; w: 3] [timeMillis: 1487946302376] [levelValue: 800] [[Message
received: Health Check notification with severity level: INFO. (host:mike-
payara, server:server, domain:domain1,instance:server) CPUC:Health Check
Result:[[status=GOOD, message='CPU%: 1.45, Time CPU used: 3 seconds 797
milliseconds'']']]]

[2017-02-24T14:25:02.380+0000] [INFO] [] [] [tid: _ThreadID=50 _ThreadName=p:
thread-pool-1; w: 5] [timeMillis: 1487946302380] [levelValue: 800] [[Message
received: Health Check notification with severity level: SEVERE. (host:mike-
payara, server:server, domain:domain1,instance:server) CPUC:Health Check
Result:[[status=CRITICAL, message='CPU%: 109.71, Time CPU used: 7
milliseconds'']']]]
```

Asadmin Commands

Set the JMS Notifier Configuration

To set the JMS notifier configuration, the following asadmin command will reflect the configuration done in the previous section:

```
asadmin set-jms-notifier-configuration --dynamic=true --enabled=true
  --contextFactoryClass=com.sun.enterprise.naming.SerialInitContextFactory
  --connectionFactoryName=jms/__defaultConnectionFactory
  --queueName=notifierQueue
  --url=localhost:7676
  --username=****
  --password=****
  --target=server-config
```

Get the JMS Notifier Configuration

To get the JMS notifier configuration using the Asadmin CLI, run the following command:

```
asadmin> get-jms-notifier-configuration

Enabled  Filter   Context Factory Class  Connection Factory Name  Queue Name  URL
Username  Password
true     WARNING  com.sun.enterprise.naming.SerialInitContextFactory
jms/__defaultConnectionFactory  notifierQueue  localhost:7676  myusername
mypassword
Command get-jms-notifier-configuration executed successfully.
```

Troubleshooting

When you have correctly configured the JMS notifier, it can be used to push notifications to your configured queue.

If you do not see any notifications, check the following:

- Is your MDB or other JMS client correctly configured to consume messages from the correct queue? (e.g. check for typos)
- Are the JMS queue details correctly set up in the notifier configuration? (check the server.log for errors)
- Is the JMS queue available? If you have configured your own JMS broker, is it responding? If the broker is remote, check that it is reachable.
- Is the service using the notifier configured to send notifications frequently enough to observe?
- Is the service using the notifier correctly configured and also enabled?

Event Bus Notifier

The Event Bus Notifier provides a way to send notifications from the notification service into the internal Payara Server event bus, which is based on Hazelcast distributed topics.

IMPORTANT	Notifications sent to the internal event bus using this notifier are intended to be received by internal server components and not by user applications. Currently, no built-in internal components consume event bus notifications yet and there are no available APIs that allow applications to consume these events.

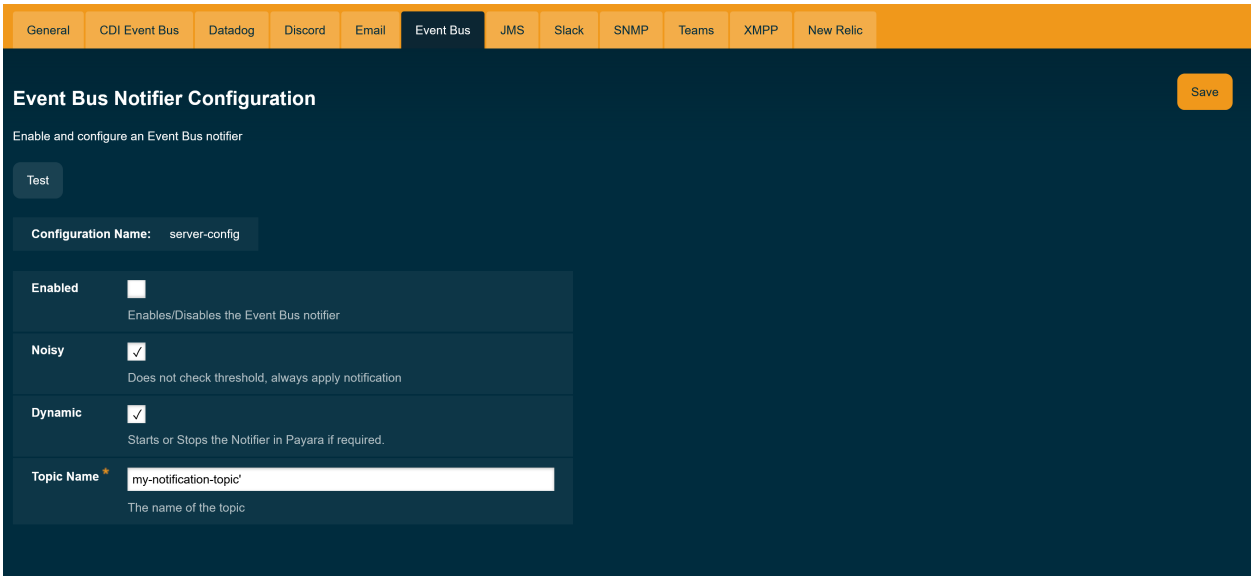
NOTE	The Data Grid must be enabled for the Event Bus to be available.

Event Bus Notifier Configuration

This notifier provides only one configuration option, the **Topic Name**, which is

mandatory.

In the Admin Console, simply set the **Topic Name** to a desired name, which will be used to set up the Hazelcast topic that will receive the notifications.



Make sure that the "Enabled" box is ticked so that the notifier will be used. If you would like the changes to take effect without needing a restart, tick the "Dynamic" box as well.

To make these changes via the Asadmin CLI, use the following subcommand:

```
asadmin set-eventbus-notifier-configuration --topicName=my-topic --dynamic=true
--enabled=true
```

To check the current applied configuration from asadmin, run the command:

```
asadmin get-eventbus-notifier-configuration

Enabled  Filter   Topic Name
false    WARNING  payara.notification.event

Command get-eventbus-notifier-configuration executed successfully.
```

CDI Event Bus Notifier

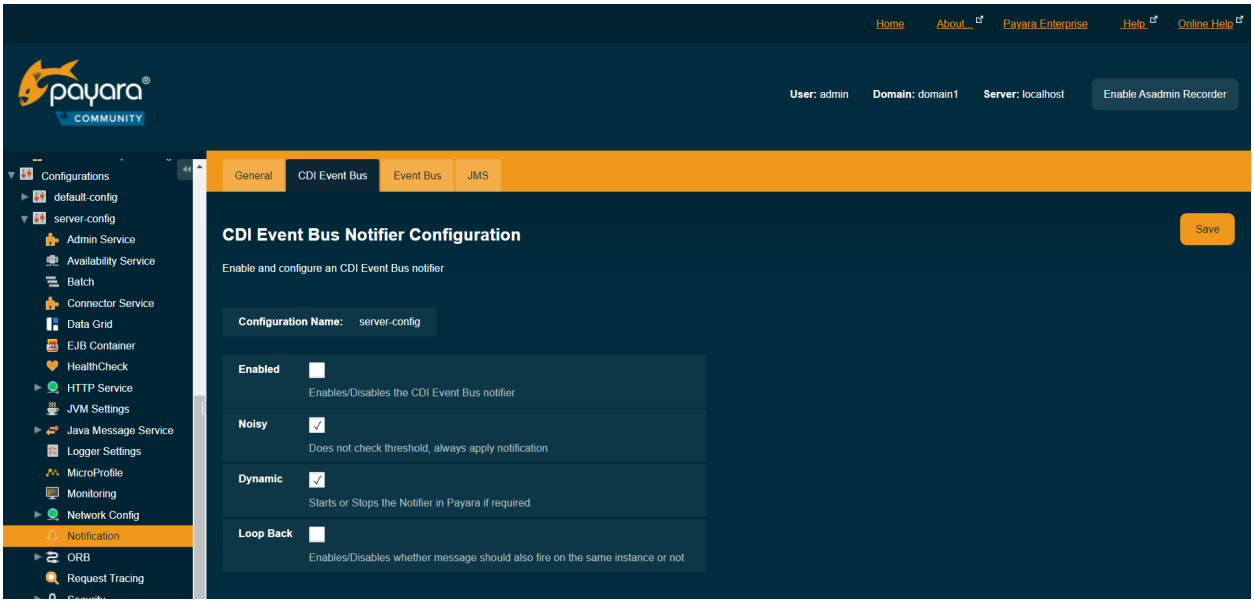
The CDI Event Bus notifier provides a way to send notifications from the notification service into the internal Payara Server's CDI Event bus.

These notifications consist of CDI asynchronous events that can be consumed by user applications with valid CDI beans set up as listeners.

NOTE	The Data Grid must be enabled for the CDI Event Bus to be available.

CDI Event Bus Notifier Configuration

You can configure the CDI Event Bus notifier from the Admin Console like this:



The following options are available:

`Enabled`: Enables/disables the notifier. `Dynamic`: Applies changes to the notifier without a server restart. `Loop Back`: Whether events should also be observed on the same instance that triggered the instance or not.

To make these changes via the Asadmin CLI, use the following subcommand:

```
asadmin> set-cdieventbus-notifier-configuration --loopBack=true --dynamic=true
--enabled=true --hazelcastEnabled=true
```

To check the current applied configuration from asadmin, run the command:

```
asadmin get-cdieventbus-notifier-configuration

Enabled  Filter   Loopback
false    WARNING  false
```

```
Command get-cdieventbus-notifier-configuration executed successfully.
```

Observing Notification Events

Any application deployed to any instance in the Data Grid can observe notification events triggered by the CDI Event bus notifier.

Event messages are instances of the `EventbusMessage` class, which provides structured data about the specific event type, such as `HealthCheckNotificationData` or `RequestTracingNotificationData`. It also provides the same information in a String form in the `title` and `message` fields.

Notification events can be observed as a standard `@Inbound` CDI event of type `EventbusMessage` or its super-types:

```
@RequestScoped
public class ObserverBean{

    public void observe(@Observes @Inbound EventbusMessage event) {
        var shortInfo = event.getSubject();
        var detailedMessage = event.getMessage();

        var domainName = event.getDomain();
        var sourceInstanceName = event.getInstance();

        if (event.getData() instanceof HealthCheckNotificationData) {
            Optional<HealthCheckResultEntry> mostCritical = event.getData()
            .as(HealthCheckNotificationData.class).getEntries()
            .stream().sorted().findFirst();
        }
    }
}
```

IMPORTANT

For an application to use the `Inbound` annotation and `EventbusMessage` class, it will have to define the "Payara Public API" as a project dependency.

Monitoring Service Notifiers Configuration

The Monitoring service can be configured to send notifications using the notification service. This allows you to forward data from exposed MBeans to your configured notifiers.

Configuration

The Monitoring service configuration page can be found under **Configurations** → `<config-name>` → **Monitoring**.

The JMX page allows you to specify the time between notifications, how notifications will be delivered, and the MBeans which will be included within the message.

Enabling Monitoring Service

To configure notification settings for JMX Monitoring, you will need to enable it. Have a look at to-enable-monitoring

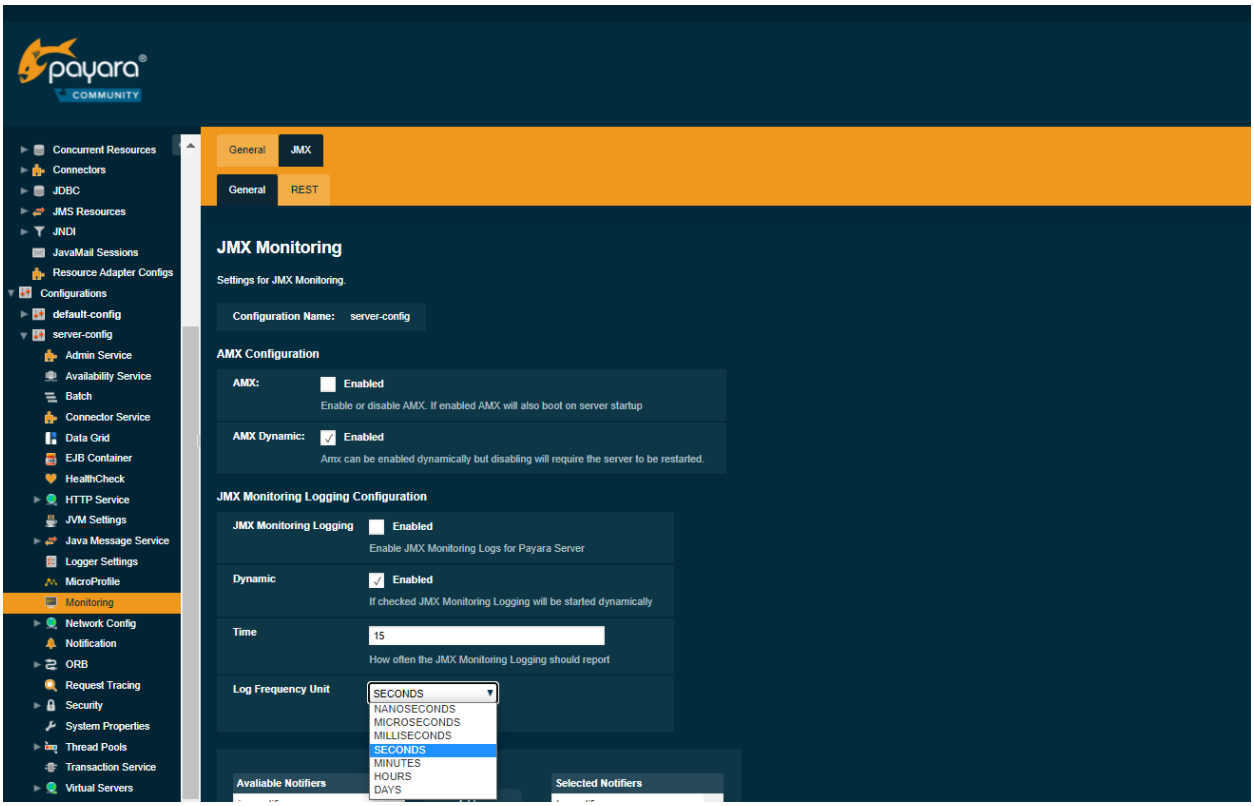
NOTE	You will need some MBean attributes configured for anything to be logged by the notifiers.

JMX monitoring notifications can be done through the Admin Console or asadmin commands.

Configuring JMX Monitoring Time via the Admin Console

To configure the frequency of logged data, enter both the intended frequency of logs and the unit of time.

For example, to log every 30 milliseconds:



Configuring JMX Monitoring Time via Asadmin CLI

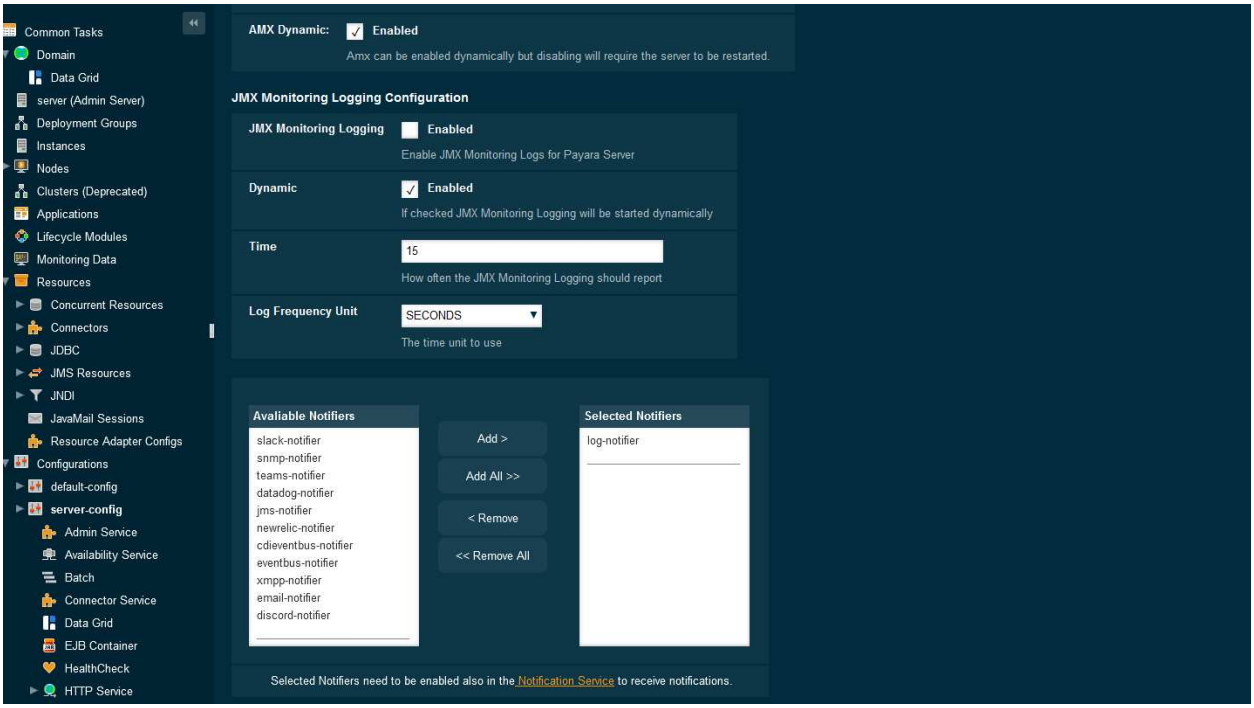
To configure the data logging frequency via asadmin, use the following command:

```
asadmin> set-jmx-monitoring-configuration --logfrequency <frequency>
--logfrequencyunit NANOSECONDS|MICROSECONDS|MILLISECONDS|SECONDS|MINUTES|HOURS|DAYS
```

Adding Notifiers via the Admin Console

Before you receive notifications from your chosen notifier, please ensure that you have configured the notification service beforehand.

To select notifiers to use for JMX Monitoring, hold the `CTRL` or `SHIFT` key to select multiple notifiers, and then click `Add >`.



Adding Notifiers via Asadmin

For each notifier you wish to enable use the following command:

```
asadmin set-jmx-monitoring-configuration --setNotifiers=log-notifier,jms-notifier,cdieventbus-notifier --dynamic=true
```

By using the `setNotifiers` option, you can set the current list of notifiers updated by the Monitoring service.

> **TIP**
>
> You can get the list of all available notifiers by using the `list-notifiers` asadmin command.

See Also

- "Logging Configuration"

Extended Notification Service Details

Payara Enterprise has additional notifiers built in and ready for use with your server environment that do not require further installation. These additional notifiers are pre-installed as **extensions** in the server module layout.

TIP

If needed, notifiers can be removed from the server's environment to reduce overhead.

Notifier Extension Structure

Each notifier extension is composed of 2 parts:

The Core Module

The core module contains all the code necessary to receive notifications from the notification service and propagate them to their proposed channel. When installing this module, you'll get the admin commands to configure the notifier, as well as the ability to select that notifier in the notification sources such as Request Tracing commands and admin console pages.

The Admin Console Plugin

The admin console plugin contains all the code necessary to configure the notifier in the admin console. This is exclusively the tab in the notification service page. This module is not required for the notifier to function, but makes administrating the notifier easier.

Installing a Notifier

To install a notifier extension: download the artifacts listed on the notifier documentation page and drop them into the `as-install/modules` directory of your Payara Server installation.

You can then restart the server to see your extension activated.

Removing Notifiers

All Payara Server notifiers are included in Payara Enterprise by default, however they are fully pluggable and can be removed from your Payara Server installation.

To remove a notifier from Payara Enterprise edition, first find the relevant modules to the notifier you want to remove in `as-install/modules`. To fully remove the notifier, you should delete both the core module and the admin console plugin module files, then restart the domain.

If you change your mind in the future, you can just drop both the module files back into the `as-install/modules` directory and restart the domain for the notifier to be re-installed.

Additional Notifiers

The following notifiers are available in Payara Server Enterprise:

Email Notifier

Payara Server is able to direct notifications to a single given email address. The notifier modules are available in these locations:

- Email Notifier Core 2.0-enterprise
- Email Notifier Console Plugin 2.0-enterprise

To use this notifier the following requirements have to be met:

- A valid, configured `JavaMail` session.

Configuration

At a high level, the steps to configure the email notifier are:

1. Create a `JavaMail` Session
2. Create the notifier using the `asadmin` command or through the Admin Console

Using the Admin Console

You will need a `JavaMail` session to send the notifications; instructions on setting up a `JavaMail` session can be found "in this section".

The email notifier is configured in the **Email** tab of the Notification Section of the server's instance configuration.

On the Email tab there are four fields:

- `Enabled`: Determines whether the notifier should be activated.
- `Dynamic`: Sets whether the notifier will be enabled without a restart, `true` enables it

immediately, while `false` requires a restart.

- `JavaMail JNDI Name`: Sets the name of the Java Naming and Directory Interface (JNDI) of the configured `JavaMail` session. The parameter name is `--jndiName` in the asadmin command.
- `Target Address`: Sets the email address which will receive the email notifications. Currently only one address can be specified. The parameter name is `--recipient` in the asadmin command.

[Email Notifier on Admin Console]

Using the Asadmin CLI

The same configuration can be set using an `asadmin` command, as shown below:

```
asadmin set-email-notifier-configuration --jndiName=mail/exampleEmailNotifier
--recipient=notifications@example.com --enabled=true --dynamic=true
```

To check the currently applied configuration, you can run the following `asadmin` command:

```
asadmin get-email-notifier-configuration
```

It will return the current configuration of the email notifier in full detail; including whether it is enabled, the recipient address, and the JNDI name of the current JavaMail session in use.

Here is an example output:

```
Enabled  JNDI Name                  Recipient
true     mail/exampleEmailNotifier  example@example.com
```

SNMP Notifier

The **S**imple **N**etwork **M**anagement **P**rotocol is a standard Internet Protocol that was designed to collect and organize information about all managed devices and services that belong to the same network. **SNMP** is supported by a huge variety of devices,

including network devices like routers, modems and switches, servers and in the latter years extended to cover IoT devices and specialized equipment connected to the cloud.

The notifier modules are available in these locations:

- SNMP Notifier Core 2.0-enterprise
- SNMP Notifier Console Plugin 2.0-enterprise
- Dependency: SNMP4J 3.7.5

IMPORTANT	Use the version of SNMP4J 3.7.5 provided above as it has a modified manifest to be compatible with Payara Enterprise.

SNMP Concepts

The basic elements in every SNMP communication are the following:

Managed Component: This is the device or service that is managed and monitored by SNMP, in this specific case, Payara Server. **SNMP Agent**: A specialized software that gathers data on the overall status of the managed component and exposes specific instructions over interfaces for its management. In our case, the SNMP Notifier acts as an SNMP agent. **SNMP Manager**: A server machine that runs a network management system, which is configured to monitor and manage all devices and services in a network by *"talking"* to the agents and gathering all relevant information.

The SNMP protocol works using *PDUs (Protocol Data Units)* to allow managers and agents to work with a defined set of instructions.

NOTE	The notifier currently operates using the **TRAP** PDU to handle notification events.

SNMP Traps

Payara Server notifies network management systems using SNMP traps. An SNMP trap is a special PDU that is used to generate an asynchronous notification from an *agent* to a *manager*. An important point to stress with traps is that they are asynchronous events

that can occur at any time, which means that a *manager* expects to receive updates in a **PASSIVE** manner, thus saving resources.

In order to correctly configure Payara Server to send traps to a manager, the following information must be reviewed in tandem with the configuration of the *manager*:

Hostname: This is the hostname of the server where the manager software resides. **Port**: This is the UDP port where the manager software is receiving SNMP traps. By default, the protocol usually uses port **162**. **Version**: The version used by the SNMP trap. **Community**: This is the community string used to "secure" the communication to the manager **OID**: The object identifier used by the trap in order to allow the manager to correctly interpret the status of the server.

Version

Currently, the SNMP protocol is supported on three versions: **v1**, **v2c** and **v3**. Version 1 is extremely unsecure since authentication is managed only with the community string. Version 2c is an enhanced version of v1 in that improves the message formats and the protocol operations, using stronger authentication guards using the simplistic community string mechanism.

Finally, Version 3 improves security and remote management discarding the use of community strings.

Currently, the notifier configuration only supports **v1** and **v2c**, hence the need to correctly configure a community string as well. By default, all traps are configured to use **v2c**, since it's guaranteed to be supported by almost all network manager systems.

Community

The community string is used to establish trust between managers and agents. The community names are essentially passwords; most vendors ship their equipment with default community strings: `public` for a read-only community (allows reading data values, but doesn't allow modification) and `private` for the read-write community (allowed to read and modify data values). It's important to change these defaults on production environments.

The notifier configures all traps with the `public` community string by default.

OID - Object Identifier

Agents expose management data as variables composed in a structured hierarchy. SNMP does not define which variables are exposed by a managed component. Instead, SNMP uses an extensible design which allows applications and services to define their own hierarchies and other metadata (such as type and description of the variable), which are described in a **Management Information Base (MIB)**.

MIBs describe the structure of the management data of a device subsystem using a hierarchical namespace composed of Object Identifiers (OID). Each OID uniquely identifies a variable sent in the SNMP trap by the agent.

In order for a **network management system** to understand a trap sent to it by an agent, the management system must know what the OID defines. Therefore, it must have the MIB for that trap loaded in its configuration.

The default value used by the notification service is `.1.3.6.1.2.1.1.8`. This OID represents a simple octet string that is used to hold the content of the notification event.

NOTE	In most cases there is no need to change the default OID, since it's verified by most of the standard MIBs included on SNMP management systems and tools.

Configuration

To set up the SNMP notifier on Payara Server you need to input the values mentioned earlier in the domain configuration. As usual, you can do this using the administration web console or from the command line.

TIP	Keep in mind that the only required value to configure the notifier is the **hostname** of your network management system. All other values will assume the defaults discussed earlier if not configured explicitly.

Using the Admin Console

To configure the Notification Service in the Administration Console, go to *Configuration* → *[instance-configuration (like server-config)]* → *Notification Service* and click on the **SNMP** tab:

[Notification Service in Admin Console]

Using the Asadmin CLI

To configure the Notification Service from the command line, use the `set-snmp-notifier-configuration` asadmin command like this:

```
asadmin set-snmp-notifier-configuration --enabled=true --dynamic=true --hostname
=localhost --snmpport=162 --community=public --oid=".1.3.6.1.2.1.1.8" --version=v2c
```

You can use the `--enabled` and `--dynamic` options to enable or disable the SNMP notifier on demand.

Also, you can retrieve the current configuration for the SNMP notifier using the `get-snmp-notifier-configuration` asadmin command like this:

```
asadmin > get-snmp-notifier-configuration
```

This will return the details of the current SNMP configuration; see below for an example:

```
Enabled  Filter   Community  OID                Version  Host       SNMP Port
true     WARNING  example    .1.3.6.1.2.1.1.8   v2c      127.0.0.1  162
```

Troubleshooting

When you have correctly configured the SNMP notifier, it can be used to push notifications to your configured server. You can visualize the notification messages on your network management system of your choice.

If you do not see any notification event messages, check the following:

- Is the SNMP notifier enabled?

- Is the Notification Service itself enabled?
- Is there a service configured to use the notifier? (e.g. the HealthCheck service)
- Is the service configured to send notifications frequently enough to observe?
- Have you enabled the service after configuring it?
- Does the SNMP network management supports SNMP traps?
- Does the SNMP network management system support the configured protocol version?
- Is the community string correctly supported by the SNMP network management system?
- Are the SNMP management system's MIB correctly configured to verify traps sent with the configured OID?
- Is there a firewall between Payara Server and the network management system that is correctly configured to allow sending SNMP traps in the respective port?

Here's a sample of how the SNMP traps are visualized using MG-Soft Trap Ringer software:

[SNMP Traps onTRinger]

New Relic Notifier

New Relic is an application performance monitoring (APM) tool that can be used to gather statistics and notify on key critical events generated by different type of applications.

New Relic specializes on integrating Java application servers like GlassFish, JBoss WildFly, etc. and other application frameworks seamlessly.

Payara Server includes a special notifier to send notifications from the "Notification service" to the New Relic **Insights service** using its remote API.

Custom notification events generated from Payara Server's own monitoring systems

like the Request Tracing and Healthcheck services can be automatically identified by this API and gathered correctly to identify performance issues on the application server.

The notifier modules are available in these locations:

- New Relic Notifier Core 2.0-enterprise
- New Relic Notifier Console Plugin 2.0-enterprise

New Relic Configuration

In order for the notifier to work correctly, you must complete the following requirements:

1. Create a New Relic user account. You can sign up for a new account here

NOTE	New Relic offers 14-days trial accounts that can be used to test the service for free.

2. Configure application performance monitoring (APM) for a Payara Server instance by installing a New Relic Java agent
3. Configure a New Relic Insights API key that will be used to set up the notifier.

Configure Application Performance Monitoring for Payara Server

The first task is to set up the New Relic Java agent in Payara Server. To do that, follow these steps:

1. Login to New Relic with your account credentials, and if you haven't set up a monitoring agent yet, the following screen will be shown:

   [New Relic agent setup welcome screen]

2. Select the **Java** option and the agent installation procedure will be shown:

   [New Relic Java agent setup procedure]

   From the screen shown, copy the value of the *License Key* (you need to press the **Reveal License Key** button first) since you will need it later.

3. Now, on the same screen press the **Download the Java Agent**. This will download the Java agent files to your station. Unzip the files and copy the resulting folder to the root folder of the Payara Server domain you want to monitor.

   In the following example we are copying it to the default domain `domain1`:

```
unzip newrelic-java-3.XX.X.zip
mv newrelic as-install/domains/domain1
```

4. Before proceeding to install the agent into the Payara Server domain, we need to configure the License Key we got earlier. To do this, modify the agent's configuration file that's located at `domain-dir/newrelic/newrelic.yml` by using the `license_key` property:

```
  # ...
  # This section is for settings common to all environments.
  # Do not add anything above this next line.
  common: &default_settings

    # ============================== LICENSE KEY ===============================
    # You must specify the license key associated with your New Relic
    # account. ...
    license_key: '073f5089681d0e96d5e0a9691251a626ea90286d'

    # Agent Enabled
    # Use this setting to disable the agent instead of removing it from the
startup command.
    # Default is true.
    agent_enabled: true

    # Set the name of your application as you'd like it to show up in New Relic.
    # ...
    app_name: Payara Server 7.2.0

    # ...
```

NOTE	We also recommend setting the name of the application under the `app_name` property to correctly identify this server's domain on the APM dashboard

IMPORTANT	The License Key value must be inserted using single quotes (')

1. Now, proceed to install the agent by running the following command at the root of the domain folder:

```
java -jar newrelic/newrelic.jar install
```

If the installation was successful, the output of the agent's installer will show it like this:

```
***** ( ( o))  New Relic Java Agent Installer
***** Installing version 3.39.1 ...

* Backed up start script to
/opt/7.2.0/glassfish/domains/domain1/config/domain.xml.20170613_180108
* Added agent switch to start script
/opt/7.2.0/glassfish/domains/domain1/config/domain.xml

* No need to create New Relic configuration file because:
 A config file already exists:
/opt/7.2.0/glassfish/domains/domain1/newrelic/newrelic.yml

***** Install successful

***** Next steps:
You\'re almost done! To see performance data for your app:

1) Restart your app server
2) Exercise your app
3) Log into http://rpm.newrelic.com

Within two minutes, your app should show up, ready to monitor and troubleshoot.
If app data doesn\'t appear, check newrelic/logs/newrelic_agent.log for errors.
```

The installer will modify the domain.xml configuration file of your server's domain and add the configuration of the Java agent as a JVM start argument:

```
<java-config classpath-suffix="" debug-options="-
agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=9009" system-
classpath="">
  <jvm-options>-
javaagent:${com.sun.aas.instanceRoot}/newrelic/newrelic.jar</jvm-options>
  <jvm-options>-server</jvm-options>
  ...
</java-config>
```

NOTE	The agent setup page has a different set of instructions for Windows and Linux environments, but they are so similar that the instructions

mentioned here should be sufficient.

TIP

If you are considering installing the New Relic Java agent on a Payara Server standalone instance or a cluster of multiple instances, you need to manually add the JVM (the installer won't work) option with the `javaagent` configuration to the instance/cluster referenced configuration.
Also, make sure that the directory where the *newrelic.jar* resides also contains the `newrelic.yml` configuration file

2. Now that the agent has been installed, start the Payara Server domain. You will observe the following messages at the start of the domain's log

```
INFO: New Relic Agent: Loading configuration file
"/opt/7.2.0/glassfish/domains/domain1/newrelic/newrelic.yml"
INFO: New Relic Agent: Writing to log file:
/opt/7.2.0/glassfish/domains/domain1/newrelic/logs/newrelic_agent.log
```

3. After some minutes with the server working, you can log in back to the New Relic portal and on the **APM** dashboard you can see a new entry for recently configured server:

   [New Relic APM Dashboard]

   You can also access the latest monitoring statistics:

   [New Relic Application Details]

Retrieving the API Key from New Relic Insights

Now that the agent has been correctly installed, login back to your New Relic account portal and head to the **Insights** dashboard:

[New Relic Insights Dashboard]

Access the *Manage Data* option on the side menu, you will be presented the following screen:

[New Relic Insights API Keys]

Click on the `+` icon at the side of the **Insert Keys** header. Take note of the *Account ID* and *API Key* value on this screen. Also add a brief description to reference this key on the dashboard:

[New Relic Insights Insert API Keys]

Notifier Configuration

With the New Relic Java agent and Insights API Key correctly configured, you can proceed to configure the New Relic notifier on the Payara Server domain. As usual, you can do this using the administration web console or from the command line.

Using the Administration Web Console

To configure the Notification Service in the Administration Console, go to *Configuration* → *[instance-configuration (like server-config)]* → *Notification Service* and click on the **New Relic** tab:

[New Relic Configuration on Admin Console]

Check the **Enabled** box (and the **Dynamic** box too if you don't want to restart the domain) and input the New Relic Account ID and the newly inserted Insights API Key. Hit the **Save** button to preserve the changes.

From the Command Line

To configure the Notification Service from the command line, use the `set-newrelic-notifier-configuration` asadmin command, specifying the tokens like this:

```
asadmin set-newrelic-notifier-configuration --dynamic=true --enabled=true
--accountId=1658989 --key=b5815wdxj6lF_tmMBljQa5y1603JTiLh
```

You can use the `--enabled` and `--dynamic` options to enable or disable the New Relic notifier on demand.

Also, you can retrieve the current configuration for the New Relic notifier using the `get-`

`newrelic-notifier-configuration` asadmin command like this:

```
asadmin get-newrelic-notifier-configuration

Enabled  Filter  Key                                Account Id
true     WARNING   b5815wdxj6lF_tmMBljQa5y1603JTiLh  1658989
```

Troubleshooting

When you have correctly configured the New Relic notifier, it can be used observe notification events on the New Relic Insights service dashboard. If you do not see any notification event messages on the data explorer, check the following:

- Is the New Relic notifier enabled?
- Is the Notification Service itself enabled?
- Is there a service configured to use the notifier? (e.g. the HealthCheck service)
- Is the service configured to send notifications frequently enough to observe?
- Have you enabled the service after configuring it?
- Is the **Java Agent** for your Payara Server's domain correctly configured?
- Have you created a valid API Key for the Insights service?
- Does your account ID and the Insights API key match the ones configured on the Payara Server notifier?

Here's a sample of how these notifications are visualized on the Data Explorer dashboard for the New Relic Insights service:

[New Relic Insights Data Explorer]

You can observe that the events are correctly identified depending on the severity of the notification, and they are correctly grouped by their custom category (in the image only the **Healthcheck** events are shown).

Datadog Notifier

Datadog is a cloud metric aggregator that simplifies the gathering and visualization of

multiple platform stacks and services that can be on premises or on the cloud. Datadog also eases browsing through of all this information by implementing combination graphs (multiple graphs that can be configured on a single view) of related metrics.

Datadog supports gathering metrics from the following sources:

- Cloud Providers (Amazon Web Services, Microsoft Azure, Google Cloud Platform)
- Message Queues (Apache Kafka, ActiveMQ, RabbitMQ)
- Code Repositories (GitHub, Bitbucket)
- Clustering and DevOps Tools (Apache Mesos, Kubernetes, Docker)
- Web Servers (Apache HTTP, Nginx, LightHTTP)
- SQL and NoSQL Databases (MySQL, PostgreSQL, MongoDB, CouchBase)
- Monitoring Solutions (New Relic, Nagios, Splunk, Amazon Cloud Trail)

Payara Server includes a special notifier to send notifications from the "Notification service" to a Datadog account by means of an API key.

IMPORTANT	Currently, the Payara Datadog notifier only supports the `US1 – East` region.

The notifier modules are available in these locations:

- Datadog Notifier Core 2.0-enterprise
- Datadog Notifier Console Plugin 2.0-enterprise

Datadog Account Integration

When you create a Datadog account, the service will provide you a default API key that you can use to integrate Payara Server with your account.

NOTE	You can sign up for a free trial account (14 days) if you want to test this integration without having to worry about a permanent subscription.

Here are the instructions to retrieve the API keys:

1. First, access the Datadog administration panel located at https://app.datadoghq.com/. Then, in the side menu select your name in the bottom left *Organisation Settings* → *API Keys* option:

   [Datadog Admin Panel Side Menu]

2. On the API Keys tab you will observe the default API Key available for integration:

   [Datadog API Key Panel]

   You can also create a new API key if you want to use it exclusively for your Payara Server installation.

3. Assign a name to the default API key or a newly created key, so you can identify which services or platform stacks are integrating with Datadog other than your Payara Server installation:

   [Datadog API Key Name]

Notifier Configuration

Having retrieved the API key from Datadog's administration panel, it's necessary to configure this key to the notification service on Payara Server. It's possible to do this either by using the administration web console or from the command line using administration commands.

Using the Administration Web Console

To configure the Notification Service in the Administration Console, go to *Configuration* → *[instance-configuration (like server-config)]* → *Notification Service* and click on the **Datadog** tab:

[Datadog Configuration on Admin Console]

Check the **Enabled** box (and the **Dynamic** box too if you don't want to restart the domain) and input the three tokens highlighted in the previous section.

Hit the **Save** button to preserve the changes.

From the Command Line

To configure the Notification Service from the command line, use the `set-datadog-notifier-configuration` asadmin command, specifying the tokens like this:

```
asadmin set-datadog-notifier-configuration --dynamic=true --key
=af510f122026bc9107f7c5f4f049e597 --enabled=true
```

You can use the `--enabled` and `--dynamic` options to enable or disable the Datadog notifier on demand.

Also, you can retrieve the current configuration for the Datadog notifier using the `get-datadog-notifier-configuration` asadmin command like this:

```
asadmin get-datadog-notifier-configuration

Enabled  Filter   Key
true     WARNING  af510f122026bc9107f7c5f4f049e597
```

Troubleshooting

When you have correctly configured the Datadog notifier, Datadog can gather relevant monitoring events from your Payara Server installation. If you do not see any notification event messages on your Datadog account, check the following:

- Is the Datadog notifier enabled?
- Is the Notification Service itself enabled?
- Is there a service configured to use the notifier? (e.g. the HealthCheck service)
- Is the service configured to send notifications frequently enough to observe?
- Have you enabled the service after configuring it?
- Does the configured API key on Payara Server match one of the API keys configured on your Datadog account?
- Is Datadog running in the US1 region?

Here's a sample of how the notifications are visualized on the Datadog event notification panel (on the side menu select the *ServiceMgmt* → *Events* option):

[Datadog Events Sample]

MS Teams Notifier

A notifier extension for publishing to a Microsoft Teams channel.

The notifier modules are available in these locations:

- Teams Notifier Core 2.0-enterprise
- Teams Notifier Console Plugin 2.0-enterprise

Notifier Configuration Configuration

Using the Administration Web Console

To configure the Notification Service in the Administration Console, go to *Configuration* → *[instance-configuration (like server-config)]* → *Notification Service* and click on the **Teams** tab:

[Notification Service in Admin Console]

From the Command Line

To configure the Notification Service from the command line, use the `set-teams-notifier-configuration` asadmin command like this:

```
asadmin set-teams-notifier-configuration --enabled=true --dynamic=true
https://outlook.office.com/webhook/xyz
```

You can use the `--enabled` and `--dynamic` options to enable or disable the Teams notifier on demand.

You can retrieve the current configuration for the Teams notifier using the `get-teams-notifier-configuration` asadmin command like this:

```
asadmin get-teams-notifier-configuration
```

This will return the details of the current Teams configuration; see below for an example:

```
Enabled  Filter   Webhook URL
true     WARNING  https://outlook.office.com/webhook/xyz
```

Slack Notifier

Slack is a functional enterprise chat and team collaboration tool inspired by IRC. It allows an organisation to create specific **domains** and discuss work topics or subjects in separate **channels** that can either be public or private.

This notifier sends notification events to an organisation's domain channel using Slack API integrations.

The notifier modules are available in these locations:

- Slack Notifier Core 2.0-enterprise
- Slack Notifier Console Plugin 2.0-enterprise

Slack Integration Configuration

In order to allow the notification service to log events on a Slack channel of your choice, you must fulfill the following requirements first:

- Have created a Slack account within your organisation's domain (for example **payara.slack.com**)
- Have a Slack account with granted permissions to create channels and configure API integrations.
- Have created a Slack channel which will be used by the notification service to report notification events.

Once you have completed these requirements, you can configure an Incoming Webhook integration for the channel.

An *Incoming Webhook* is the simplest integration mechanism that Slack provides, in that

they make use of normal HTTP requests with a *JSON* payload that includes the message text and additional options to format incoming messages to the channel.

Configure Incoming Webhook

1. First, access the developer portal's application directory located at https://payara.slack.com/apps.

   Then, in the search box type **Incoming Webhooks**. A suggestion will appear in the results display:

   [Slack Application Portal]

2. Now, proceed to select the channel that the WebHook will use to direct the notification events to:

   [Incoming WebHook Channel Selection]

NOTE	It's recommended to choose a *private* channel, since it will keep these notifications events protected in case of unwanted disclosure, and it won't spam other users of your organisation with unwanted notifications

3. With this, the WebHook configuration is finished. Now, take special attention to the WebHook URL, which is usually follows this structure:

   ```
   https://hooks.slack.com/services/{TOKEN1}/{TOKEN2}/{TOKEN3}
   ```

   Like in the following sample:

   [WebHook URL Sample]

   We will use these 3 tokens when configuring the server's integration later.

   We recommend that you configure the following settings in the channel integration to give better visualization for your notifications:

Descriptive Label: Use a name that's easy to understand to identify this WebHook integration **Customize Name**: Insert a name for the "agent" that will report the notification messages **Customize Icon**: Insert a custom icon to identify the agent that will be visualized in the messages. We recommend using one of the Payara logos like in this sample:

[WebHook Additional Settings]

Notifier Configuration

With the Incoming WebHook properly configured, now it's time to set up the Notification Service in the domain configuration. As usual, you can do this using the administration web console or from the command line.

Using the Administration Web Console

To configure the Notification Service in the Administration Console, go to *Configuration* → *[instance-configuration (like server-config)]* → *Notification Service* and click on the **Slack** tab:

[Slack Configuration on Admin Console]

Check the **Enabled** box (and the **Dynamic** box too if you don't want to restart the domain) and input the three tokens highlighted in the previous section. Hit the **Save** button to preserve the changes.

From the Command Line

To configure the Notification Service from the command line, use the `set-slack-notifier-configuration` asadmin command, specifying the tokens like this:

```
asadmin set-slack-notifier-configuration --enabled=true --dynamic=true --token1
=T43CHENUC --token2=B441EFQB1 --token3=Jas8P3jCZ4wK8Y2m88dCxGkn
```

You can use the `--enabled` and `--dynamic` options to enable or disable the Slack notifier on demand.

Also, you can retrieve the current configuration for the Slack notifier using the `get-`

`slack-notifier-configuration` asadmin command like this:

```
asadmin get-slack-notifier-configuration

Enabled  Filter   Token 1    Token 2    Token 3
true     WARNING  T43CHENUC  B441EFQB1  Jas8P3jCZ4wK8Y2m88dCxGkn
```

Troubleshooting

When you have correctly configured the Slack notifier, it can be used to push notifications to your configured channel. If you do not see any notification event messages, check the following:

- Is the Slack notifier enabled?
- Is the Notification Service itself enabled?
- Is there a service configured to use the notifier? (e.g. the HealthCheck service)
- Is the service configured to send notifications frequently enough to observe?
- Have you enabled the service after configuring it?
- Is the **Incoming WebHook** correctly configured in the Slack API portal?
- Check that you have not surpassed the current rate limit for the Slack API

Here's a sample of how the notifications are visualized on the Slack Desktop application:

[Slack Notifications Sample]

Discord Notifier

A notifier for publishing notifications to a Discord channel.

The notifier modules are available in these locations:

- Discord Notifier Core 2.0-enterprise
- Discord Notifier Console Plugin 2.0-enterprise

Notifier Configuration

Using the Administration Web Console

To configure the Notification Service in the Administration Console, go to *Configuration* → *[instance-configuration (like server-config)]* → *Notification Service* and click on the **Discord** tab:

[Notification Service in Admin Console]

From the Command Line

To configure the Notification Service from the command line, use the `set-discord-notifier-configuration` asadmin command like this:

```
asadmin set-discord-notifier-configuration --enabled=true --dynamic=true
--webhookId=xxx --webhookToken=yyy
```

Where the `xxx` and `yyy` are respectively the ID and token for the webhook you have configured within Discord.

You can use the `--enabled` and `--dynamic` options to enable or disable the Discord notifier on demand.

You can retrieve the current configuration for the Discord notifier using the `get-discord-notifier-configuration` asadmin command like this:

```
asadmin get-discord-notifier-configuration
```

This will return the details of the current Discord configuration; see below for an example:

```
Enabled  Filter   Webhook ID  Webhook Token
true     WARNING  xxx         yyy
```

Administering Batch Jobs

This chapter provides procedures for administering batch jobs in the Payara Server environment by using the `asadmin` command-line utility.

Instructions for accomplishing these tasks by using the Administration Console are contained in the Administration Console online help.

About Batch Jobs

Payara Server provides a batch runtime for the scheduling and execution of batch jobs. Batch jobs are typically long-running, bulk-oriented tasks that contain a series of steps and can be executed without user interaction. Examples include billing, report generation, data format conversion, and image processing.

Batch applications submit jobs to the batch runtime and provide instructions about how and when to execute the steps. The batch runtime processes the steps as directed by job XML documents packaged with the applications and stores information about jobs in a job repository. In Payara Server, the job repository is a database.

For detailed information about batch jobs, batch processing, and the batch processing framework, see "Batch Processing" in The Jakarta Batch specifications. The specification defines the programming model for batch applications and the runtime for scheduling and executing batch jobs.

Viewing Batch Jobs

You can view detailed information about batch jobs, executions, and steps. Users who log in to the `asadmin` utility or to the Administration Console as administrator are the only users who can view details for all batch jobs submitted by all applications in the Payara Server environment.

To List Batch Jobs

Use the `list-batch-jobs` subcommand in remote mode to list batch jobs and job details.

1. Ensure that the server is running.
   Remote subcommands require a running server.
2. List batch jobs by using the "list-batch-jobs" subcommand.

To List Batch Jobs

This example lists batch jobs for the default server instance, `server`. Use `list-batch-jobs -l` to list additional details.

```
asadmin> list-batch-jobs
JOBNAME  INSTANCECOUNT
payroll  9
bonus    6
Command list-batch-jobs executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-batch-jobs` at the command line.

To List Batch Job Executions

When the batch runtime executes a job, the execution is given a unique execution ID. An execution ID is similar to a process ID. A new execution is created the first time a job is started and every time the existing execution is restarted.

Use the `list-batch-job-executions` subcommand in remote mode to list batch job executions and execution details.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List batch job executions by using the "list-batch-job-executions" subcommand.

To List Batch Jobs Executions

This example lists batch job executions for the default server instance, `server`, and displays specific details. Use `list-batch-job-executions -l` to list additional details.

```
asadmin> list-batch-job-executions -o=jobname,executionid,batchstatus,exitstatus
JOBNAME  EXECUTIONID  BATCHSTATUS  EXITSTATUS
payroll  9            COMPLETED    COMPLETED
bonus    6            FAILED       FAILED
Command list-batch-job-executions executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-batch-job-executions` at the command line.

To List Batch Job Steps

A batch job consists of one or more steps. A step is an independent and sequential phase of a batch job.

Use the `list-batch-job-steps` subcommand in remote mode to list steps and step details for a specific batch job execution.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the execution ID of an execution by using the `list-batch-job-executions` subcommand.
3. List steps for a specific batch job execution by using the "list-batch-job-steps" subcommand.

To List Batch Jobs Steps

This example lists batch job steps and specific step details for a job execution with the execution ID of `7`. The target is the default server instance, `server`. Use `list-batch-job-steps -l` to list additional details.

Some lines of output are omitted from this example for readability.

```
asadmin> list-batch-job-steps o=stepname,stepid,batchstatus,stepmetrics 7
STEPNAME   STEPID   BATCHSTATUS   STEPMETRICS
prepare    7        COMPLETED     METRICNAME          VALUE
                                  READ_COUNT          8
                                  WRITE_COUNT         8
                                  PROCESS_SKIP_COUNT  0
process    8        COMPLETED     METRICNAME          VALUE
                                  READ_COUNT          8
                                  WRITE_COUNT         8
                                  PROCESS_SKIP_COUNT  0
...
Command list-batch-job-steps executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-batch-job-steps` at the command line.

Configuring the Batch Runtime

The batch runtime uses a data source and a managed executor service to execute batch jobs. The data source stores information about current and past jobs, and the managed

executor service provides threads to jobs. Batch runtime configuration data is stored in the `config` element in `domain.xml`.

Payara Server provides a default data source and managed executor service for the execution of batch jobs. For the domain administration server (DAS), the default data source is `jdbc/TimerPool` and the default managed executor service is `concurrent/defaultManagedExecutorService`. If you create a standalone server instance or a standalone cluster, the default data source is `jdbc/__default`. You can configure the batch runtime to use different resources.

For more information about data sources, see administering-database-connectivity. For more information about managed executor services, see configuring-managed-executor-services.

To List the Batch Runtime Configuration

Use the `list-batch-runtime-configuration` subcommand in remote mode to display the configuration of the batch runtime.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Display the configuration of the batch runtime by using the "list-batch-runtime-configuration" subcommand.
3. If desired, use the `get` subcommand to view the attributes of the data source and managed executor service resources. For example (output omitted):

```
asadmin> get resources.jdbc-resource.jdbc/__TimerPool.*
...
asadmin> get resources.managed-executor-
service.concurrent/__defaultManagedExecutorService.*
...
```

To List the Batch Job Runtime Configuration

This example lists the configuration of the batch runtime for the default server instance, `server`.

```
asadmin> list-batch-runtime-configuration
DATASOURCELOOKUPNAME     EXECUTORSERVICELOOKUPNAME
```

```
jdbc/__TimerPool         concurrent/__defaultManagedExecutorService
Command list-batch-runtime-configuration executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-batch-runtime-configuration` at the command line.

To Configure the Batch Runtime

Use the `set-batch-runtime-configuration` subcommand in remote mode to configure the batch runtime.

NOTE

Do not change the data source after the first batch job has been submitted to the batch runtime for execution. If the data source must be changed, stop and restart the domain and then make the change before any jobs are started or restarted. However, once the data source has been changed, information stored in the previous data source becomes inaccessible.

The managed executor service can be changed after a batch job has been submitted to the batch runtime without affecting execution of the job.

1. Ensure that the server is running.

   Remote subcommands require a running server.

2. Configure the batch runtime by using the "set-batch-runtime-configuration" subcommand.

To Set Batch Runtime Configuration

This example configures the batch runtime for the default server instance, `server`, to use an existing managed executor service named `concurrent/Executor1`.

```
asadmin> set-batch-runtime-configuration --executorservicelookupname
concurrent/Executor1
Command set-batch-runtime-configuration executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help set-batch-runtime-configuration` at the command line.

Defining a Schema Name

Payara Server allows the ability to define the name of the *database schema* that will hold all batch job tables. This can be set via the Admin Console or using Asadmin CLI commands.

Defining a Schema Name through the Admin Console

1. Click on the instance or cluster to move to its configuration page.
2. Select the *Batch* tab, and from there click on the *Configuration* sub-tab.
3. Enter your desired value in the *Database Schema Name* field.
4. Save your changes

Defining a Schema Name Using the Asadmin CLI

It's possible to set the schema using the `set-batch-runtime-configuration` command.

The command requires you to specify the `Executor` or `DataSource` lookup name, which you can do with the `-x` or `-d` options respectively.

The command defaults to targeting the Admin Server instance (`server`), to target a different instance or cluster, use the `--target` option.

To specify the schema name, use the `--schemaName` option, or its shortcut `-n`.

An example can be seen below:

```
asadmin set-batch-runtime-configuration -d jdbc/__default --target cluster1 -n test
```

Specifying a Blank Schema Name

If you specify a blank schema name, then the schema name will depend on what database vendor is being used:

- On **MySQL**, the *test* schema will be used
- On **H2**, **Derby**, **Oracle**, **DB2** and **PostgreSQL** the schema name will be the username of the JDBC connection pool resource associated with the JDBC resource to which

JBatch is configured to use.

NOTE	This behavior only applies if you explicitly specify the schema name as blank; the schema name will still default to `APP` if not overwritten.

Setting a Table Prefix And/or Suffix

Payara Server allows you to set the prefix and/or the suffix of the batch table name. This can be set via the Admin Console, or using Asadmin CLI commands.

IMPORTANT	The table prefix and suffix settings may be ignored by non-RDBMS based datastores.

Setting a Table Prefix And/or Suffix on the Admin Console

1. Click on the instance or cluster to move to its configuration page.
2. Select the *Batch* tab, and the *Configuration* sub-tab should load.
3. Enter your desired values in the *Table Prefix* and *Table Suffix* fields.
4. Save your changes

Setting a Table Prefix And/or Suffix Using the Asadmin CLI

It's possible to set the table prefix and/or suffix is set using the `set-batch-runtime-configuration` command. The command requires you to specify the `Executor` or `DataSource` lookup name, which you can do with the `-x` or `-d` options respectively.

The command defaults to targeting the Admin Server instance (`server`), to target a different instance or cluster, use the `--target` option.

- To specify the prefix, use the `--tablePrefix` option.
- To specify the table suffix, use the `--tableSuffix` option.

An example can be seen below:

```
asadmin set-batch-runtime-configuration -d jdbc/__default --target cluster1
--tablePrefix PRE --tableSuffix SFX
```

Database Support

You can configure any of the following RDBMS engines in the same way that you would configure it to use H2 (the default database engine used to manage the JBatch data sources):

- MySQL
- PostgreSQL
- Oracle
- IBM DB2

Configuration

On the Admin console:

1. Create a Connection Pool:
    a. Navigate to *Resources* → *JDBC* → *JDBC Connection Pools* and click on *New…*
    b. Give it a name in the *Pool Name* field, select the resource type from the *Resource Type* drop-down, and choose the *Database Driver Vendor* as either **DB2**, **MySQL**, **Oracle**, or **PostgreSQL** from the *Database Driver Vendor* menu.
    c. Set any further configuration options on the next page.
    d. Click *Finish*
2. Create a JDBC Resource:
    a. Navigate to *Resources* → *JDBC* → *JDBC Resources* and click on *New…*
    b. Give it a name in the *JNDI Name* field, and select the data source you just created from the *Pool Name* drop-down.
    c. Add any additional properties and select the targets for it to be enabled on.
    d. Click *OK*.
3. Navigate to the Batch configuration page of the instance or cluster:
4. Click on the instance or cluster to move to its configuration page.

5. Select the *Batch* tab, and the *Configuration* sub-tab should load.
6. Select the new data source from the *Data Source Lookup Name* drop-down menu.
7. Save the changes.

Usage Restrictions

The Jakarta Batch runtime will not create internally more than one set of JBatch tables per schema. So in your schema if there exists a set of JBatch tables with prefixes and suffixes in the table names and then specify in the JBatch configuration for the same schema above that you wish to use JBatch tables under a different name (for example no table prefix and suffix) then during the JBatch initialization phase, JBatch will attempt to create these tables since they do not exist.

However since the table constraint names already exist for the existing JBatch tables in the same schema, table creation will fail. One can of course run a_SQL_ script to create the relevant JBatch schema objects under different names.

Mysql Specifics

For MySQL (5/8) database use, it is recommended the following additional property be set:

Name	Value	Description
`zeroDateTimeBehavior`	`convertToNull`	Action for `DATETIME` values that are composed entirely of zeros (used by MySQL to represent invalid dates)

IMPORTANT	Depending on the version of MySQL you may also need to set the server property `sql_mode` to blank as JBatch uses `null` and zero dates.

Administering Database Connectivity

This chapter provides procedures for performing database connectivity tasks in the Payara Server environment.

About Database Connectivity

A database management system (DBMS) provides facilities for storing, organizing, and retrieving data. The information in databases is often described as persistent data because it is saved on disk and exists after the application process ends. Most business applications store data in relational databases. Applications can access database information by using the Java Database Connectivity (JDBC) API.

The key elements of database connectivity are the following:

- Database. The repository where data is stored for an enterprise. Jakarta EE applications access relational databases through the JDBC API. For administration procedures, see setting-up-the-database.
- JDBC Connection Pool. A JDBC connection pool is a group of reusable connections for a particular database. For administration procedures, see administering-jdbc-connection-pools.
- JDBC Resource. A JDBC resource (data source) provides applications with a means of connecting to a database. To create a JDBC resource, specify the connection pool with which it is associated. Multiple JDBC resources can specify a single connection pool. A JDBC resource is identified by its Java Naming and Directory Interface (JNDI) name. For administration procedures, see administering-jdbc-resources.
- JDBC Driver. A database driver is a software component that enables a Java application to interact with a database connectivity API . Each database requires its own driver. For administration procedures, see integrating-the-jdbc-driver.

At runtime, the following sequence occurs when an application connects to a database:

1. The application gets the JDBC resource associated with the database by making a call through the JNDI API.
   Using the JNDI name of the resource, the naming and directory service locates the JDBC resource. Each JDBC resource specifies a connection pool.
2. Using the JDBC resource, the application gets a database connection.
   Payara Server retrieves a physical connection from the connection pool that

corresponds to the database. The pool defines connection attributes such as the database name (URL), username, and password.

3. After the database connection is established, the application can read, modify, and add data to the database.
   The application accesses the database by making calls to the JDBC API. The JDBC driver translates the application's JDBC calls into the protocol of the database server.
4. When the application is finished accessing the database, the application closes the connection and returns the connection to the connection pool.

Setting up the Database

Most applications use relational databases to store, organize, and retrieve data. Applications access relational databases through the Java Database Connectivity (JDBC) API.

To Install the Database and Database Driver

1. Install a compatible JDBC relational database product.
2. Install a supported JDBC driver for the database product.

   For a list of drivers supported by Payara Server, see configuration-specifics-for-jdbc-drivers.

3. Make the JDBC driver JAR file accessible to the domain administration server (DAS).

   See integrating-the-jdbc-driver.

4. Create the database. The application provider usually delivers scripts for creating and populating the database.

Next Steps

You are now ready to create a connection pool for the database, and a JDBC resource that points to the connection pool. See to-create-a-jdbc-connection-pool and to-create-a-jdbc-resource. The final step is to integrate the JDBC driver into an administrative domain as described in integrating-the-jdbc-driver.

To Start the Database

Payara Server includes an implementation of the H2 Database, however, you can use any JDBC-compliant database. The database is not started automatically when you start Payara Server, so if you have applications that require a database, you need to start H2 database manually by using the local `start-database` subcommand.

Start the database by using the "`start-database`" subcommand.

When the database server starts, or a client connects to it successfully, the following files are created at the location that is specified by the `--dbhome` option:

- The `h2.log` file contains the database server process log along with its standard output and standard error information.
- The database files contain your schema (for example, database tables).

Example 11-1 to Start a Database

This example starts the H2 database on the host host1 and port 5001.

```
asadmin> start-database --dbhost=localhost --dbport=5001
Starting database in Network Server mode on host localhost and port 5001.
tcp://192.168.0.110:5001
Starting database in the background.
Log redirected to C:\payara-server\_WORKDIR\7.2025.1\payara7\glassfish\databases
\h2.log.
Command start-database executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help start-database` at the command line.

To Stop the Database

Use the local `stop-database` subcommand to stop the H2 database on a specified port. A single host can have multiple database server processes running on different ports.

1. If necessary, notify users that the database is being stopped.
2. Stop the database by using the "`stop-database`" subcommand.

Example 11-2 to Stop a Database

This example stops the H2 database on port 5001 of `localhost`.

```
asadmin> stop-database --dbhost=localhost --dbport=5001
Command stop-database executed successfully.
```

Troubleshooting

For a laptop that roams between networks, you might have trouble shutting down the database. If you start the H2 database and then change your IP address, you will not be able to stop the H2 database unless you add a specific `--dbhost` argument. For example, if you run `asadmin start-database dbhost = 0.0.0.0`, and then disconnect Ethernet and switch to Wi-Fi, you should run a command similar to the following to stop the database:

```
asadmin> stop-database dbhost localhost
```

You can also view the full syntax and options of the subcommand by typing `asadmin help stop-database` at the command line.

H2 Database Documentation

For further details on how to configure the embedded H2 database you can visit the following: H2 Database Documentation.

Configuring Access to the Database

After establishing the database, you are ready to set up access for Payara Server applications. The high-level steps include creating a JDBC connection pool, creating a JDBC resource for the connection pool, and integrating a JDBC driver into an administrative domain.

Administering JDBC Connection Pools

A JDBC connection pool is a group of reusable connections for a particular database. Because creating each new physical connection is time-consuming, Payara Server maintains a pool of available connections. When an application requests a connection, it obtains one from the pool. When an application closes a connection, the connection is

returned to the pool. JDBC connection pools can be globally accessible or be scoped to an enterprise application, web module, EJB module, connector module or application client module, as described in "application-scoped-resources" in the Payara Server Application Deployment section.

A JDBC resource is created by specifying the connection pool with which the resource is associated. Multiple JDBC resources can specify a single connection pool. The properties of connection pools can vary with different database vendors. Some common properties are the database name (URL), the username, and the password.

To Create a JDBC Connection Pool

It is possible to create a JDBC connection pool using administration commands or on the web administration console.

Using Administration Commands

Use the `create-jdbc-connection-pool` subcommand in remote mode to register a new JDBC connection pool with the specified JDBC connection pool name. A JDBC connection pool or a connector connection pool can be created with authentication. You can either use a subcommand option to specify user, password, or other connection information using the `asadmin` utility, or specify the connection information in the XML descriptor file.

One connection pool is needed for each database, possibly more depending on the application. When you are building the connection pool, certain data specific to the JDBC driver and the database vendor is required. You can find some of the following specifics in configuration-specifics-for-jdbc-drivers:

- Database vendor name
- Resource type, such as `javax.sql.DataSource` (local transactions only) `javax.sql.XADataSource` (global transactions)
- Data source class name
- Required properties, such as the database name (URL), username, and password

Creating a JDBC connection pool is a dynamic event and does not require server restart. However, there are some parameters that do require server restart. See configuration-changes-that-require-restart.

Before You Begin

Before creating the connection pool, you must first install and integrate the database and its associated JDBC driver. For instructions, see setting-up-the-database.

1. Ensure that the server is running.
   Remote subcommands require a running server.
2. Create the JDBC connection pool by using the create-jdbc-connection-pool subcommand.
3. If needed, restart the server. Some parameters require server restart. See configuration-changes-that-require-restart.

Example 11-3 to Create a JDBC Connection Pool

This example creates a JDBC connection pool named `sample_derby_pool` on `localhost`.

```
asadmin> create-jdbc-connection-pool --datasourceclassname
org.apache.derby.jdbc.ClientDataSource --restype javax.sql.XADataSource --property
portNumber=1527:password=APP:user=APP:serverName=localhost:databaseName=sun-
appserv-samples:connectionAttributes=\;create\\=true sample_derby_pool

Command create-jdbc-connection-pool executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-jdbc-connection-pool` at the command line.

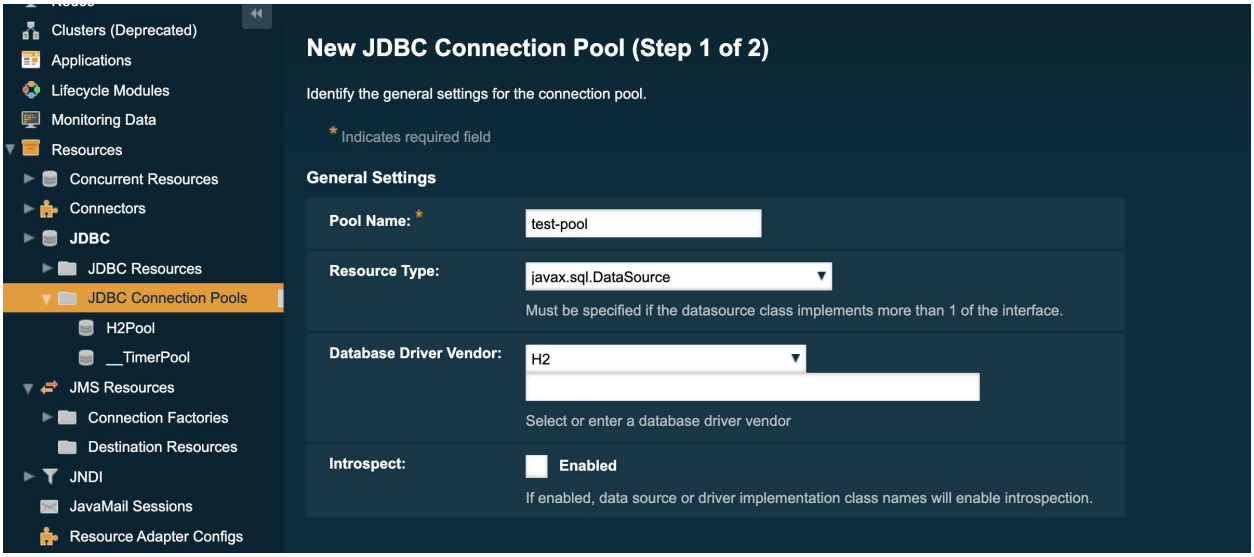
Using Admin Console

In the left hand panel of the Admin console go to *Resources* → *JDBC* → *JDBC Connection Pools*

Click the *New* button and enter the following values:

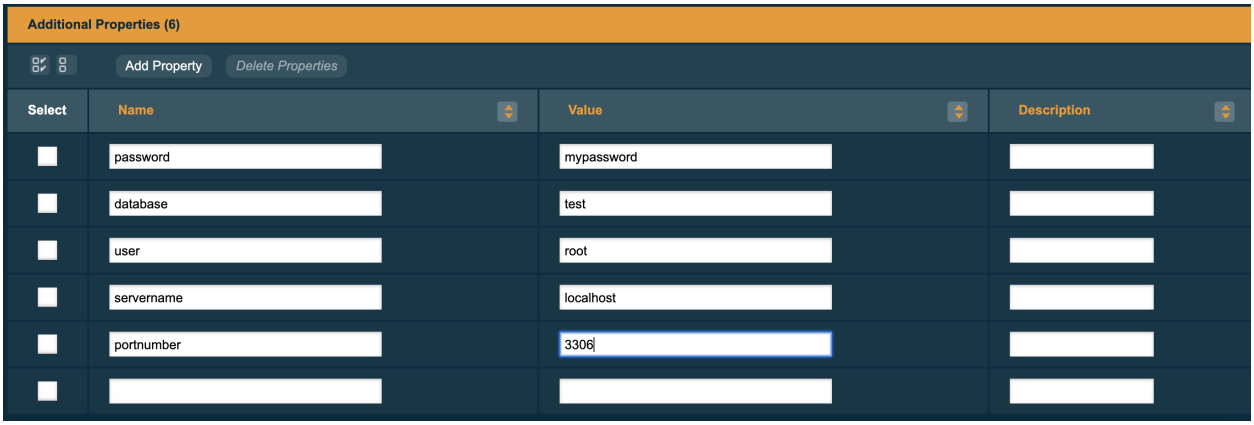
Setting Name	Value
Pool Name	test-pool
Resource Type	`javax.sql.DataSource`
Driver Vendor	(value for the driver, e.g. Derby30 for a Derby DB, MySQL for a MySQL DB)

Click Next and then click Finish on the next screen, accepting the default values.



Add the following properties to the connection pool by going to the *Additional Properties* tab of the connection pool:

Setting name	Value	Description
PortNumber	3306	Database port
Password	`mypassword`	Database password
User	root	Database user
serverName	localhost	Database address
DatabaseName	test	Name of the database
connectionAttributes		Other connection attributes (supported by the JDBC driver)



To test the connection attributes, go back to the *General* tab and click the *Ping* button at the top of the screen. You should see a message stating that the ping succeeded.

To List JDBC Connection Pools

Use the `list-jdbc-connection-pools` subcommand in remote mode to list all existing JDBC connection pools.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the JDBC connection pools by using the "`list-jdbc-connection-pools`" subcommand.

Example 11-4 to List a JDBC Connection Pool

This example lists the JDBC connection pools that are on `localhost`.

```
asadmin> list-jdbc-connection-pools
sample_derby_pool2
poolA
__TimerPool
H2Pool
sample_derby_pool
Command list-jdbc-connection-pools executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-jdbc-connection-pools` at the command line.

To Contact (Ping) a Connection Pool

Use the `ping-connection-pool` subcommand in remote mode to test if a connection pool is usable. For example, if you create a new JDBC connection pool for

an application that is expected to be deployed later, you can test the JDBC pool with this subcommand before the application is deployed.

TIP	Running a ping will force the creation of the pool if it hasn't already been created.

Before You Begin

Before you can contact a connection pool, the connection pool must be created with authentication, and the server or database must be running.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Ping a connection pool by using the "`ping-connection-pool`" subcommand.

Example 11-5 to Contact (Ping) a Connection Pool

This example tests to see if the `DerbyPool` connection pool is usable.

```
asadmin> ping-connection-pool DerbyPool
Command ping-connection-pool executed successfully
```

You can also view the full syntax and options of the subcommand by typing `asadmin help ping-connection-pool` at the command line.

You can also specify that a JDBC connection pool is automatically tested when created or reconfigured by setting its `--ping` option to `true`(the default is `false`). See to-create-a-jdbc-connection-pool or to-update-a-jdbc-connection-pool.

To Reset (Flush) a Connection Pool

Use the `flush-connection-pool` in remote mode to reinitialize all connections established in the specified connection pool without the need for reconfiguring the pool. Connection pool reconfiguration can result in application redeployment, which is a time-consuming operation.

The JDBC connection pool or connector connection pool is reset to its initial state. Any existing live connections are destroyed, which means that the transactions associated

with these connections are lost and must be retried. The subcommand then recreates the initial connections for the pool, and restores the pool to its steady pool size.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Reset a connection pool by using the `flush-connection-pool` subcommand.

Example 11-6 Resetting (Flushing) a Connection Pool

This example resets the JDBC connection pool named `__TimerPool` to its steady pool size.

```
asadmin> flush-connection-pool __TimerPool
Command flush-connection-pool executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help flush-connection-pool` at the command line.

To Update a JDBC Connection Pool

You can change all settings for an existing pool except its name. Use the `get` and `set` subcommands to view and change the values of the JDBC connection pool properties.

1. List the JDBC connection pools by using the "`list-jdbc-connection-pools`" subcommand.
2. View the attributes of the JDBC connection pool by using the get subcommand.

   For example: `asadmin get resources.jdbc-connection-pool.H2Pool.property`
3. Set the attribute of the JDBC connection pool by using the `set` subcommand.

   For example: `asadmin set resources.jdbc-connection-pool.H2Pool.steady-pool-size=9`
4. If needed, restart the server.

Some parameters require server restart. See configuration-changes-that-require-restart.

Connection Pool Sizing

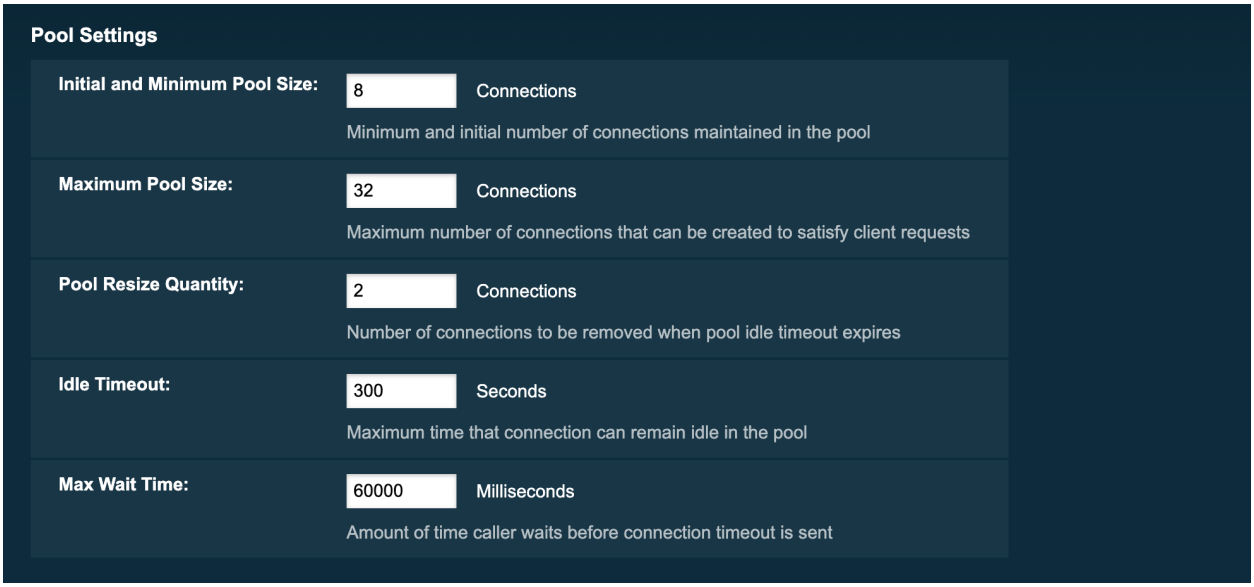
Connection pools should be sized to cater for the maximum number of concurrent connections.

The maximum size should be set in accordance with the maximum number of client requests your system or application can process. If your application receives 100 requests and each of those requires a database connection then if your connection pool is anything less than 100 some of those requests will have to wait for a connection to either be created or become available.

The minimum size of the connection pool ensures that a number of connections to the database are always established - this means that if you have a pool with a minimum size of 10, and you receive 10 requests then all can retrieve a database connection without waiting for the pool to create a new connection.

Setting the Sizes Using the Admin Console

Click on the connection pool name and under the *Pool Settings* section you will find Initial and Minimum Pool Size and Maximum Pool Size. Set these to your required sizes:



Setting the Sizes Using Administration Commands

To set the initial and minimum pool size (to the value of **10** for example), use the following administration commands:

```
asadmin> set resources.jdbc-connection-pool.test-pool.steady-pool-size=10
```

To set the maximum pool size (to the value of **200** for example), use the following administration commands:

```
asadmin> set resources.jdbc-connection-pool.test-pool.max-pool-size=200
```

Connection Validation

Connection validation ensures that connections aren't assigned to your application after the connection has already gone stale.

Connection validation is always a trade-off between how sure you want to be that a connection is valid and the performance impact that the validation mechanism can exert into application that use the pool. There will be a negative performance impact by having to return an invalid connection by your application and borrow a new one, so finding the right balance is key.

Before using a connection from the pool a simple query is sent to test the connection. If there is an issue with the connection, it is removed from the pool and another one is used. If you have an issue such as the database being down, and you have a large number of connections then each of those connections will be tested and removed.

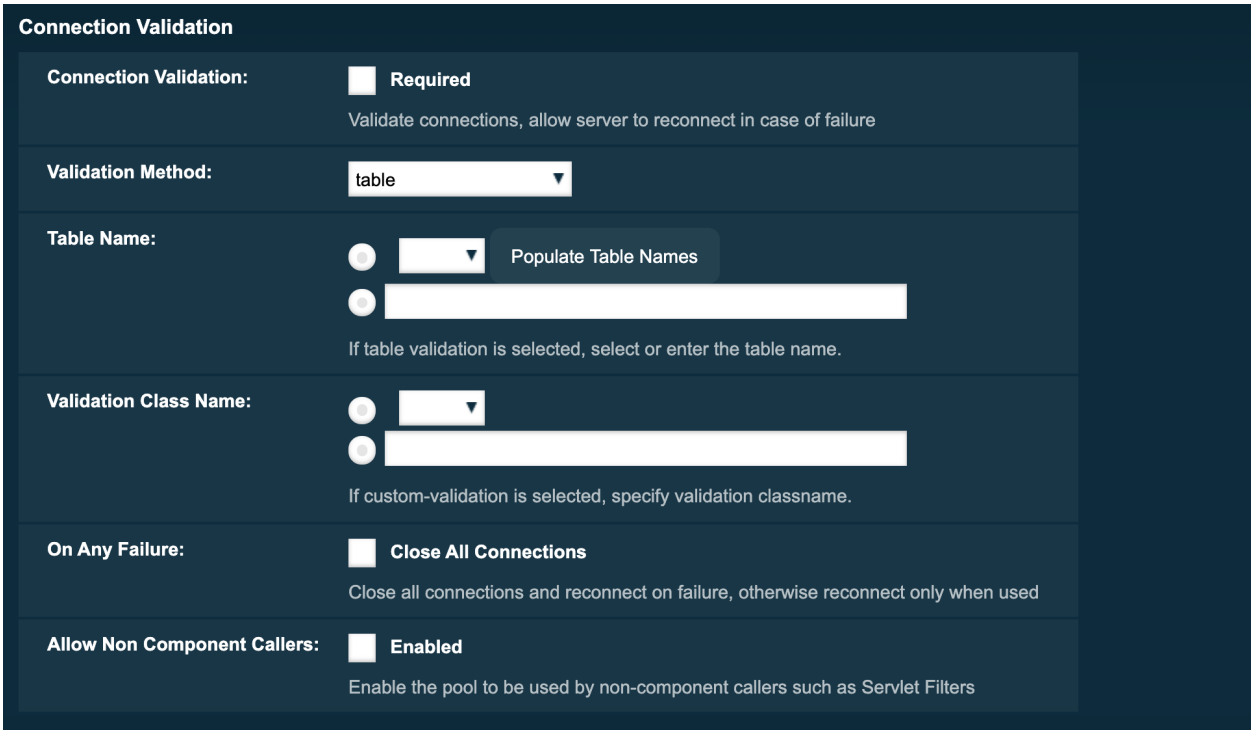
In order to avoid this, you can set a connection validation so that if a connection fails, all remaining open connections are closed.

Set up the Validation Using the Admin Console

1. Click on the name of the connection pool
2. Select the *Advanced* tab
3. Scroll down to the *Connection Validation* section and select the following settings:
    a. *Connection Validation required*
    b. *Validation method* - custom-validation
    c. *Validation class name* – a proper class name for the chosen JDBC driver, such as

`org.glassfish.api.jdbc.validation.DerbyConnectionValidation`
for a **Derby DB**

From the same screen, you can also set whether to close all connections on failure.



Additional information regarding enabling connection validation through 'asadmin' commands, along with various options, is elaborated in the the "`Handling Invalid Connections`" section of our Payara Server Application Development JDBC documentation.

To Delete a JDBC Connection Pool

Use the `delete-jdbc-connection-pool` subcommand in remote mode to delete an existing JDBC connection pool. Deleting a JDBC connection pool is a dynamic event and does not require server restart.

Before You Begin

Before deleting a JDBC connection pool, all associations to the resource must be removed.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the JDBC connection pools by using the "`list-jdbc-connection-pools`" subcommand.
3. If necessary, notify users that the JDBC connection pool is being deleted.
4. Delete the connection pool by using the "`delete-jdbc-connection-pool`" subcommand.

Example 11-7 to Delete a JDBC Connection Pool

This example deletes the JDBC connection pool named `H2Pool`.

```
asadmin> delete-jdbc-connection-pool jdbc/H2Pool
Command delete-jdbc-connection-pool executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-jdbc-connection-pool` at the command line.

Configuring Specific JDBC Connection Pool Features

In Payara Server, JDBC Connection Pools support a variety of features to simplify administration, monitoring and performance tuning.

Transparent Pool Reconfiguration

When the properties or attributes of a JDBC connection pool are changed, the connection pool is destroyed and re-created. Normally, applications using the connection pool must be redeployed as a consequence. This restriction can be avoided by enabling transparent JDBC connection pool reconfiguration. When this feature is enabled, applications do not need to be redeployed. Instead, requests for new connections are blocked until the reconfiguration operation completes. Connection requests from any in-flight transactions are served using the old pool configuration to complete the transaction. Then, connections are created using the pool's new configuration, and any blocked connection requests are served with connections from the re-created pool.

To enable transparent JDBC connection pool reconfiguration, set the `dynamic-`

`reconfiguration-wait-timeout-in-seconds` property of the JDBC connection pool to a positive, nonzero value in one of the following ways:

- Add it as a property in the Edit JDBC Connection Pool Properties page in the Administration Console.
- Specify it using the `--property` option in the `create-jdbc-connection-pool` subcommand. For more information, see "`create-jdbc-connection-pool`".
- Set it using the `set` subcommand. For example:

```
asadmin set resources.jdbc-connection-pool.pool-name.property.dynamic-
reconfiguration-wait-timeout-in-seconds=15
```

This property specifies the time in seconds to wait for in-use connections to close and in-flight transactions to complete. Any connections in use or transaction in flight past this time must be retried.

Using an Initialization Statement

You can specify a statement that executes each time a physical connection to the database is created (not reused) from a JDBC connection pool. This is useful for setting request or session specific properties and is suited for homogeneous requests in a single application. Set the *Init SQL* attribute of the JDBC connection pool to the SQL string to be executed in one of the following ways:

- Enter an *Init SQL* value in the Edit Connection Pool Advanced Attributes page in the Administration Console.
- Specify the `--initsql` option in the `asadmin create-jdbc-connection-pool` command. For more information, see "`create-jdbc-connection-pool`".
- Specify the `init-sql` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.H2Pool.init-sql="sql-string"
```

Setting a Statement Timeout

An abnormally long-running JDBC query executed by an application may leave it in a

hanging state unless a timeout is explicitly set on the statement. Setting a statement timeout guarantees that all queries automatically time out if not completed within the specified period.

When statements are created, the `queryTimeout` is set according to the statement timeout setting. This works only when the underlying JDBC driver supports `queryTimeout` for `Statement`, `PreparedStatement`, `CallableStatement`, and `ResultSet`.

You can specify a statement timeout in the following ways:

- Enter a Statement Timeout value in the Edit Connection Pool Advanced Attributes page in the Administration Console.
- Specify the `--statementtimeout` option in the `asadmin create-jdbc-connection-pool` command. For more information, see "`create-jdbc-connection-pool`".

Statement and Connection Leak Detection and Reclamation

This feature allows you to set specific time-outs so that if SQL statements or JDBC connections haven't been closed by an application (potentially leading to a memory leak) they can be logged and/or closed.

If statements are not closed by an application after use, it is possible for the application to run out of cursors. Enabling statement leak detection causes statements to be considered as leaked if they are not closed within a specified period. Additionally, leaked statements can be reclaimed automatically.

By default, these values are set to **0** meaning this detection feature is turned off.

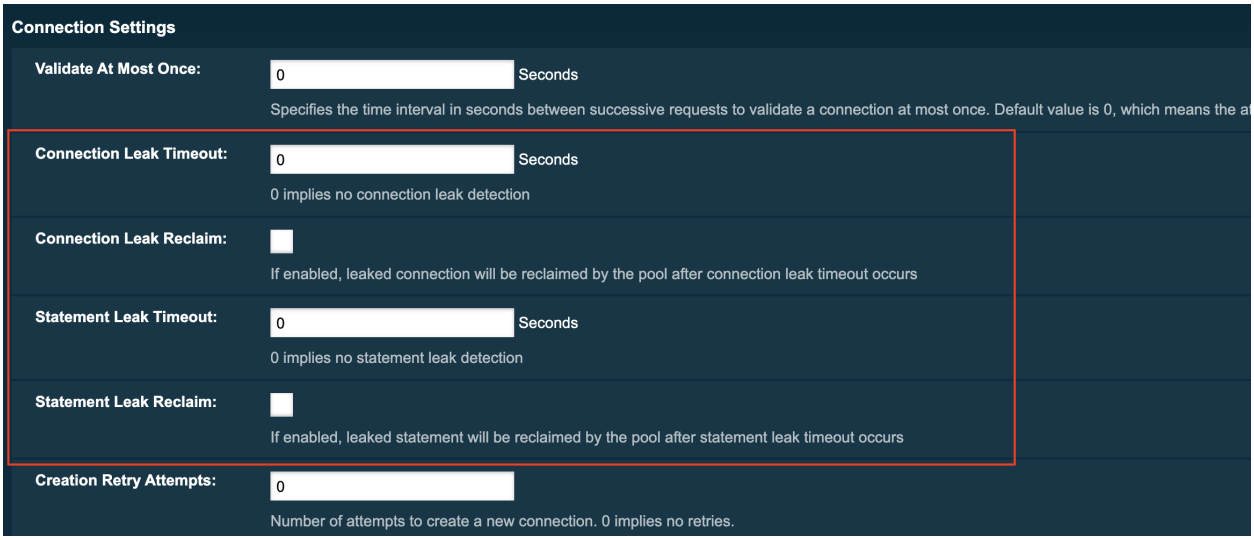
TIP	When working on a test or staging environment, it is recommended that leaks are logged after a short timeout but not closed. On production environments, it is recommended that leaks are closed and all logged leaks are monitored instead.

When selecting a value for Statement Leak Timeout In Seconds, make sure that:

- It is less than the Connection Leak Timeout; otherwise, the connection could be closed before the statement leak is recognized.
- It is greater than the Statement Timeout; otherwise, a long-running query could be mistaken as a statement leak.

Configuring Leak Detection Using the Admin Console

1. Click on the name of the JDBC connections pool
2. Select the *Advanced* tab
3. Scroll down to *Connection Settings*
4. Set the *Connection Leak Timeout* and *Statement Leak Timeout* value in seconds



Configuring Leak Detection Using Administration Commands

You also can set the time-out values using the following `asadmin` commands:

```
asadmin> set resources.jdbc-connection-pool.test-pool.statement-leak-timeout-in-
seconds=5
asadmin> set resources.jdbc-connection-pool.test-pool.connection-leak-timeout-in-
seconds=5
```

You can turn on reclaiming of the leaking resources with the following commands:

```
asadmin> set resources.jdbc-connection-pool.DerbyPool.connection-leak-reclaim=true
```

```
asadmin> setresources.jdbc-connection-pool.DerbyPool.statement-leak-reclaim=true
```

Once these values are set, if connection or statement leaks are detected, you will see messages similar to the example below in the application log:

```
WARNING: A potential connection leak detected for connection pool test-pool. The
stack trace of the thread is provided below:
 ...
```

Enabling JDBC Call Logging

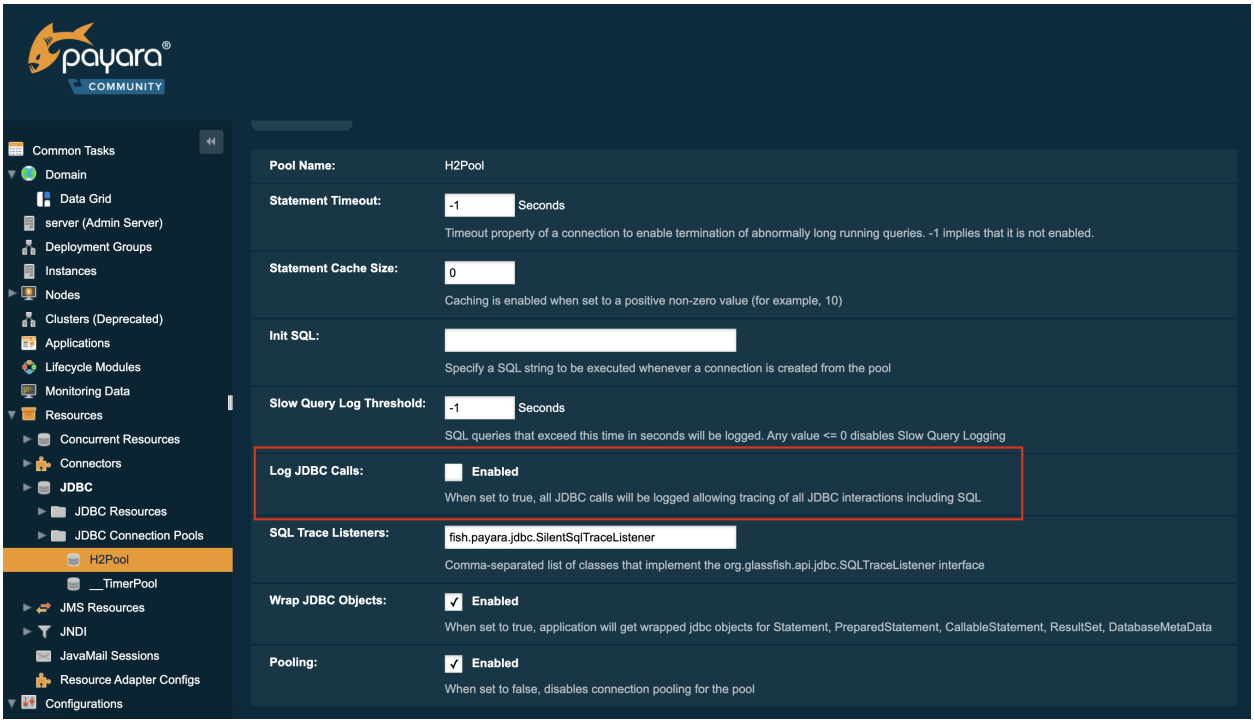
With **Log JDBC Calls** configured on the connection call each call into the connection pool is timed and logged to the server log at the `FINE` level. A typical log message with Log JDBC Enabled is shown below:

```
[#|2025-11-12T18:51:01.467+0000|FINE|Payara
7.2025.1|javax.enterprise.resource.sqltrace.com.sun.gjc.util|_ThreadID=35;_ThreadNa
me=http-listener-
1(5);_TimeMillis=1454611861467;_LevelValue=500;ClassName=com.sun.gjc.util.SQLTraceL
ogger;MethodName=sqlTrace;|
  PoolName=H2Pool | ExecutionTime=1ms |
ClassName=org.apache.derby.client.net.NetConnection40 | MethodName=prepareStatement
| arg[0]=SELECT ID, AGE, BIO, BIRTHDATE, BIRTHDAY, DATEFORMAT, DATEOFBIRTH,
DATEOFHIRE, EMAIL, HIREDATE, HIREDAY, MEMBERAGE, NAME, TODAYSDATE FROM MEMBERENTITY
WHERE (NAME = ?) | arg[1]=1003 | arg[2]=1007 | |#]
```

Using the Administration Console

SQL Tracing can be enabled through the Payara Server Web administration console. First, navigate to the advanced table of your JDBC connection pool, then select the *JDBC > JDBC Connection pools > ${Your Connection pool}*.

Then select the **Advanced Tab** in the main window of the administration console. Then select the checkbox next to **Log JDBC Calls** to enable logging of all SQL calls:



Using Asadmin Commands

The **Log JDBC Calls** setting for a JDBC connection pool can also be configured via asadmin using the `set` command to set the `fish.payara.log-jdbc-calls` property of your connection pool to `true`, like in the following example:

```
asadmin set domain.resources.jdbc-connection-pool.__TimerPool.log-jdbc-calls=true
```

Advanced Connection Pool Properties in Deployment Descriptors

Payara Server supports setting advanced JDBC connection pool properties directly on data source definitions made with the `@DataSourceDefinition` annotation or directly on deployment descriptors.

Setting Properties in the Deployment Descriptor

In a Jakarta EE application, a data source definition can be added to a deployment descriptor of an application. For a web application, you would this in the standard *web.xml* deployment descriptor. To set advanced properties just add a `property` element as shown in the following example:

```
<data-source>
    <name>java:global/ExampleDataSource</name>
```

```
    <class-name>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</class-name>
    <server-name>localhost</server-name>
    <port-number>3306</port-number>
    <database-name>mysql</database-name>
    <user>test</user>
    <password>test</password>
    <!-- Example of how to use a Payara specific custom connection pool setting -->
    <property>
       <name>fish.payara.is-connection-validation-required</name>
       <value>true</value>
    </property>
</data-source>
```

Setting Properties on the `@DataSourceDefinition` Annotation.

In Jakarta EE applications, a JDBC data source can be deployed by adding the `@DataSourceDefinition` annotation to a managed component (like and `@Stateless` EJB bean for example). To configure advanced JDBC properties, use the `properties` attribute of the annotation to define an array of the properties to configure:

```
@DataSourceDefinition(
    name = "java:app/MyApp/MyDS",
    className = "org.h2.jdbcx.JdbcDataSource",
    url = "jdbc:h2:mem:test",
    properties = {"fish.payara.is-connection-validation-required=true"})
public class ApplicationComponent{
}
```

Full List of Properties

The complete list of configurable properties is the following:

Property	Value Type	Default	Notes
`fish.payara.is-connection-validation-required`	`Boolean`	`false`	`true` - Validate connections, allow server to reconnect in case of failure
`fish.payara.connection-validation-method`	`String`		The method of connection validation table, `auto-commit`, `meta-data`, `custom-validation`

Property	Value Type	Default	Notes
`fish.payara.validation-table-name`	`String`		The name of the table used for validation if the validation method is set to `table`
`fish.payara.validation-classname`	`String`		The name of the custom class used for validation if the `validation-method` is set to `custom-validation`
`fish.payara.fail-all-connections`	`Boolean`	`false`	Close all connections and reconnect on failure, otherwise reconnect only when used
`fish.payara.allow-non-component-callers`	`Boolean`	`false`	Enable the pool to be used by non-component callers such as Servlet Filters
`fish.payara.validate-atmost-once-period-in-seconds`	`Number`	`0`	Specifies the time interval in seconds between successive requests to validate a connection at most once. Default value is `0`, which means the attribute is not enabled.
`fish.payara.connection-leak-timeout-in-seconds`	`Number`	`0`	`0` implies no connection leak detection
`fish.payara.connection-leak-reclaim`	`Boolean`	`false`	If enabled, leaked connection will be reclaimed by the pool after connection leak timeout occurs

Property	Value Type	Default	Notes
`fish.payara.connection-creation-retry-attempts`	`Number`	`0`	Number of attempts to create a new connection. 0 implies no retries
`fish.payara.connection-creation-retry-interval-in-seconds`	`Number`	`10`	Time interval between retries while attempting to create a connection. Effective when Creation Retry Attempts is greater than 0.
`fish.payara.statement-timeout-in-seconds`	`Number`	`-1`	Timeout property of a connection to enable termination of abnormally long-running queries. `-1` implies that it is not enabled.
`fish.payara.lazy-connection-enlistment`	`Boolean`	`false`	Enlist a resource to the transaction only when it is actually used in a method
`fish.payara.lazy-connection-association`	`Boolean`	`false`	Connections are lazily associated when an operation is performed on them
`fish.payara.associate-with-thread`	`Boolean`	`false`	When the same thread is in need of a connection, it can reuse the connection already associated with that thread
`fish.payara.pooling`	`Boolean`	`true`	When set to `false`, disables connection pooling for the pool

Property	Value Type	Default	Notes
`fish.payara.statement-cache-size`	`Number`	`0`	Caching is enabled when set to a positive non-zero value (for example, `10`)
`fish.payara.match-connections`	`Boolean`	`true`	Turns connection matching for the pool ON or OFF
`fish.payara.max-connection-usage-count`	`Number`	`0`	Connections will be reused by the pool for the specified number of times, after which they will be closed. `0` implies the feature is not enabled.
`fish.payara.wrap-jdbc-objects`	`Boolean`	`true`	When set to `true`, application will get wrapped jdbc objects for `Statement`, `PreparedStatement`, `CallableStatement`, `ResultSet`, `DatabaseMetaData`
`fish.payara.sql-trace-listeners`	`String`		Comma-separated list of classes that implement the `org.glassfish.api.jdbc.SQLTraceListener` interface

Property	Value Type	Default	Notes
`fish.payara.ping`	`Boolean`	`false`	When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes
`fish.payara.init-sql`	`String`		Specify a SQL string to be executed whenever a connection is created from the pool
`fish.payara.statement-leak-timeout-in-seconds`	`Number`	`0`	`0` implies no statement leak detection
`fish.payara.statement-leak-reclaim`	`Boolean`	`false`	If enabled, leaked statement will be reclaimed by the pool after statement leak timeout occurs
`fish.payara.statement-cache-type`	`String`		
`fish.payara.slow-query-threshold-in-seconds`	`Number`	`-1`	SQL queries that exceed this time in seconds will be logged. Any value ⩽ 0 disables Slow Query Logging
`fish.payara.log-jdbc-calls`	`Boolean`	`false`	When set to `true`, all JDBC calls will be logged allowing tracing of all JDBC interactions including SQL
`fish.payara.skip-client-info-validation`	`Boolean`	`false`	When set to `true`, client info will not be validated.

Example Data Source Configuration

An example data source configured on the *web.xml* and deployed with a custom **SQL Trace Listener** is shown below. This data source is configured to also validate all connections returned from the connection pool before giving them to the application using the built-in *MySQL* **Connection Validation** class. The data source is also configured to log any queries that exceed **5** seconds and also logs ALL JDBC statements:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
     version="3.1">
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
    <data-source>
      <name>java:global/ExampleDataSource</name>
      <class-name>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</class-name>
      <server-name>localhost</server-name>
      <port-number>3306</port-number>
      <database-name>mysql</database-name>
      <user>test</user>
      <password>test</password>
      <!-- Example of how to use a Payara specific custom connection pool setting
-->
      <property>
          <name>fish.payara.slow-query-threshold-in-seconds</name>
          <value>5</value>
      </property>
      <property>
          <name>fish.payara.log-jdbc-calls</name>
          <value>true</value>
      </property>
      <property>
          <name>fish.payara.sql-trace-listeners </name>

<value>fish.payara.examples.payaramicro.datasource.example.CustomSQLTracer</value>
      </property>
      <property>
          <name>fish.payara.is-connection-validation-required</name>
          <value>true</value>
      </property>
      <property>
          <name>fish.payara.connection-validation-method</name>
          <value>custom-validation</value>
      </property>
      <property>
          <name>fish.payara.validation-classname</name>
          <value>
org.glassfish.api.jdbc.validation.MySQLConnectionValidation</value>
      </property>
```

```
    </data-source>
</web-app>
```

Administering JDBC Resources

A JDBC resource, also known as a data source, provides an application with a means of connecting to a database. Typically, you create a JDBC resource for each database that is accessed by the applications deployed in a domain. Multiple JDBC resources can be specified for a database. JDBC resources can be globally accessible or be scoped to an enterprise application, web module, EJB module, connector module or application client module, as described in "application-scoped-resources" in the Payara Server Application Deployment section.

A JDBC resource is created by specifying the connection pool with which the resource will be associated. Use a unique Java Naming and Directory Interface (JNDI) name to identify the resource. For example, the JNDI name for the resource of a payroll database might be `java:comp/env/jdbc/payrolldb`.

The Jakarta EE standard specifies that certain default resources be made available to applications, and defines specific JNDI names for these default resources. Payara Server makes these names available through the use of logical JNDI names, which map Jakarta EE standard JNDI names to specific Payara Server resources. For JDBC resources, the Jakarta EE standard name `java:comp/DefaultDataSource` is mapped to the `jdbc/__default` resource.

To Create a JDBC Resource

Use the `create-jdbc-resource` subcommand in remote mode to create a JDBC resource. Creating a JDBC resource is a dynamic event and does not require server restart.

Because all JNDI names are in the `java:comp/env` subcontext, when specifying the JNDI name of a JDBC resource in the Administration Console, use only the `jdbc/`name format. For example, a payrolldatabase might be specified as `jdbc/payrolldb`.

Before You Begin

Before creating a JDBC resource, you must first create a JDBC connection pool. For instructions, see to-create-a-jdbc-connection-pool.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create a JDBC resource by using the "`create-jdbc-resource`" subcommand.

   Information about properties for the subcommand is included in this help page.
3. If necessary, notify users that the new resource has been created.

Example 11-8 to Create a JDBC Resource

This example creates a JDBC resource named `H2Pool`.

```
asadmin> create-jdbc-resource --connectionpoolid H2Pool jdbc/H2Pool
Command create-jdbc-resource executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-jdbc-resource` at the command line.

To List JDBC Resources

Use the `list-jdbc-resources` subcommand in remote mode to list the existing JDBC resources.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List JDBC resources by using the "`list-jdbc-resources`" subcommand.

Example 11-9 to List a JDBC Resource

This example lists JDBC resources for `localhost`.

```
asadmin> list-jdbc-resources
jdbc/__TimerPool
jdbc/H2Pool
jdbc/__default
jdbc1
Command list-jdbc-resources executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-jdbc-resources` at the command line.

To Update a JDBC Resource

You can enable or disable a JDBC resource by using the `set` subcommand. The JDBC resource is identified by its dotted name.

1. List JDBC resources by using the "`list-jdbc-resources`" subcommand.
2. Modify the values for the specified JDBC resource by using the "`set`" subcommand.

For example:

Example 11-10 to Update a JDBC Resource

This example changes the `res1` enabled setting to `false`.

```
asadmin>set resources.jdbc-resource.res1.enabled=false
```

To Delete a JDBC Resource

Use the `delete-jdbc-resource` subcommand in remote mode to delete an existing JDBC resource. Deleting a JDBC resource is a dynamic event and does not require server restart.

Before You Begin

Before deleting a JDBC resource, all associations with this resource must be removed.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List JDBC resources by using the "`list-jdbc-resources`" subcommand.
3. If necessary, notify users that the JDBC resource is being deleted.
4. Delete a JDBC resource by using the "`delete-jdbc-resource`" subcommand.

Example 11-11 to Delete a JDBC Resource

This example deletes a JDBC resource named `H2Pool`.

```
asadmin> delete-jdbc-resource jdbc/H2Pool
Command delete-jdbc-resource executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-jdbc-resource` at the command line.

Enabling the `jdbc/__default` Resource in a Clustered Environment

Payara Server includes a preconfigured JDBC resource with the JNDI name `jdbc/default`. This `jdbc/default` resource is not enabled by default in a traditional cluster, so you need to explicitly enable it if you want to use it in a cluster.

To Enable the `jdbc/__default` Resource for a Clustered Environment

Instructions for creating JDBC resources in general are provided in to-create-a-jdbc-resource. Use the following procedure to enable the preconfigured `jdbc/__default` resource for a clustered Payara Server environment.

1. Create the `jdbc/default` resource reference for the cluster. for example: 'asadmin create-resource-ref --target cluster-name jdbc/default'
2. Enable the resource on the DAS that manages the cluster, for example: 'asadmin set resources.jdbc-connection-pool.DerbyPool.property.serverName=DAS-machine-name'

This step is only required if the cluster includes remote instances. Restart the DAS and the target cluster(s).

```
asadmin stop-cluster cluster-name
asadmin stop-domain domain-name
asadmin start-domain domain-name
asadmin start-cluster cluster-name
```

Integrating the JDBC Driver

To use JDBC features, you must choose a JDBC driver to work with Payara Server, then you must set up the driver.

Supported Database Drivers

For a list of the JDBC drivers currently supported by the Payara Server, see the Payara Release Notes. For configurations of supported and other drivers, see configuration-specifics-for-jdbc-drivers.

NOTE	Because the drivers and databases supported by the Payara Server are constantly being updated, and because database vendors continue to upgrade their products, always check with Oracle technical support for the latest database support information.

Making the JDBC Driver JAR Files Accessible

To integrate the JDBC driver into a Payara Server domain, copy the JAR files into the `domain-dir/lib` directory, then restart the server. This makes classes accessible to all applications or modules deployed on servers that share the same configuration. For more information about Payara Server class loaders, see ""Class Loaders"" in the Payara Server Application Development section.

Automatic Detection of Installed Drivers

The Administration Console detects installed JDBC Drivers automatically when you create a JDBC connection pool. To create a JDBC connection pool using the Administration Console, open the Resources component, open the JDBC component, select Connection Pools, and click on the New button. This displays the New JDBC Connection Pool page.

Based on the Resource Type and Database Vendor you select on the New JDBC Connection Pool page, data source or driver implementation class names are listed in the Datasource Classname or Driver Classname field when you click on the Next button. When you choose a specific implementation class name on the next page, additional properties relevant to the installed JDBC driver are displayed in the Additional Properties section.

Configuration Specifics for JDBC Drivers

Payara Server is designed to support connectivity to any database management system

by using a corresponding JDBC driver.

IBM DB2 Database Type 4 Driver

The JAR file for the DB2 driver is `db2jcc.jar`. Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: DB2
- DataSource Classname: `com.ibm.db2.jcc.DB2SimpleDataSource`
- Properties:
  - `databaseName` - Set as appropriate.
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.
  - `driverType` - Set to `4`.

Apache Derby Db/derby Type 4 Driver

The JAR file for the Apache Derby DB driver is `derbyclient.jar`. Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: Apache Derby
- DataSource Classname: Specify one of the following:

```
org.apache.derby.jdbc.ClientDataSource40
org.apache.derby.jdbc.ClientXADataSource40
```

- Properties:
  - `serverName` - Specify the host name or IP address of the database server.

- `portNumber` - Specify the port number of the database server if it is different from the default.
- `databaseName` - Specify the name of the database.
- `user` - Specify the database user.
  This is only necessary if the Apache Derby database is configured to use authentication. The Apache Derby database does not use authentication by default. When the user is provided, it is the name of the schema where the tables reside.
- `password` - Specify the database password.
  This is only necessary if the Apache Derby database is configured to use authentication.

Mysql Server Database Type 4 Driver

The JAR file for the MySQL driver is `mysql-connector-java-5.1.14-bin.jar`.
Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: MySql
- DataSource Classname:

```
com.mysql.jdbc.jdbc2.optional.MysqlDataSource
com.mysql.jdbc.jdbc2.optional.MysqlXADataSource
```

- Properties:
  - `serverName` - Specify the host name or IP address of the database server.
  - `portNumber` - Specify the port number of the database server.
  - `databaseName` - Set as appropriate.
  - `user` - Set as appropriate.

- `password` - Set as appropriate.

Oracle 10 Database Driver

The JAR file for the Oracle 10 database driver is `ojdbc14.jar`. Make sure that the shared library is available through `LD_LIBRARY_PATH` and that the `ORACLE_HOME` property is set.

To make the Oracle driver behave in a Jakarta EE-compliant manner, you must define the following JVM property:

```
-Doracle.jdbc.J2EE13Compliant=true
```

Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: Oracle
- DataSource Classname: Specify one of the following:

```
oracle.jdbc.pool.OracleDataSource
oracle.jdbc.xa.client.OracleXADataSource
```

- Properties:
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.

Oracle 11 Database Driver

The JAR file for the Oracle 11 database driver is `ojdbc6.jar`.

To make the Oracle driver behave in a Jakarta EE-compliant manner, you must define the following JVM property:

```
-Doracle.jdbc.J2EE13Compliant=true
```

Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: Oracle
- DataSource Classname: Specify one of the following:

```
oracle.jdbc.pool.OracleDataSource
oracle.jdbc.xa.client.OracleXADataSource
```

- Properties:
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.

NOTE

For this driver, the `XAResource.recover` method repeatedly returns the same set of in-doubt Xids regardless of the input flag. According to the XA specifications, the Transaction Manager initially calls this method with `TMSTARTSCAN` and then with `TMNOFLAGS` repeatedly until no Xids are returned. The `XAResource.commit` method also has some issues. To disable this Payara Server workaround, the `oracle-xa-recovery-workaround` property value must be set to `false`.
Additionally, in order for the transaction manager to recover transactions, the JDBC connection pool's database user must be given certain Oracle permissions:
SELECT permission on DBA_PENDING_TRANSACTIONS, PENDING_TRANS$, DBA_2PC_PENDING and DBA_2PC_NEIGHBORS.
EXECUTE permissions on DBMS_XA and DBMS_SYSTEM.

Postgresql Type 4 Driver

The JAR file for the PostgreSQL driver is `postgresql-9.0-801.jdbc4.jar`.
Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: Postgresql
- DataSource Classname: `org.postgresql.ds.PGSimpleDataSource`
- Properties:
  - `serverName` - Specify the host name or IP address of the database server.
  - `portNumber` - Specify the port number of the database server.
  - `databaseName` - Set as appropriate.
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.

Datadirect Type 4 Driver for IBM DB2 Database

The JAR file for DataDirect driver is `db2.jar`. Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: DataDirect-DB2
- DataSource Classname: `com.ddtek.jdbcx.db2.DB2DataSource`
- Properties:
  - `serverName` - Specify the host name or IP address of the database server.
  - `portNumber` - Specify the port number of the database server.
  - `databaseName` - Set as appropriate.
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.

Datadirect Type 4 Driver for IBM Informix

Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: DataDirect-Informix
- DataSource Classname: Specify one of the following:

```
com.informix.jdbcx.IfxDataSource
com.informix.jdbcx.IfxXADataSource
```

  DataDirect DataSource Classname:
  `com.ddtek.jdbcx.informix.InformixDataSourcee`

- Properties:
  - `serverName` - Specify the Informix database server name.
  - `portNumber` - Specify the port number of the database server.
  - `databaseName` - Set as appropriate. This is optional.
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.
  - `IfxIFXHost` - Specify the host name or IP address of the database server.

Datadirect Type 4 Driver for Microsoft SQL Server Database

The JAR file for the DataDirect driver is `sqlserver.jar`. Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: DataDirect-Microsoft SQL Server
- DataSource Classname:
  `com.ddtek.jdbcx.sqlserver.SQLServerDataSource`

- Properties:
  - `serverName` - Specify the host name or IP address and the port of the database server.
  - `portNumber` - Specify the port number of the database server.
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.
  - `selectMethod` - Set to `cursor`.

Datadirect Type 4 Driver for Mysql Server Database

The JAR file for the DataDirect driver is `mysql.jar`. Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: DataDirect-MySQL
- DataSource: `com.ddtek.jdbcx.mysql.MySQLDataSource`
- Properties:
  - `serverName` - Specify the host name or IP address and the port of the database server.
  - `portNumber` - Specify the port number of the database server.
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.
  - `selectMethod` - Set to `cursor`.

Datadirect Type 4 Driver for Oracle 11 Database

The JAR file for the DataDirect driver is `oracle.jar`.

To make the Oracle driver behave in a Jakata EE-compliant manner, you must define the following JVM property:

```
-Doracle.jdbc.J2EE13Compliant=true
```

Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: DataDirect-Oracle
- DataSource Classname: `com.ddtek.jdbcx.oracle.OracleDataSource`
- Properties:
  - `serverName` - Specify the host name or IP address of the database server.
  - `portNumber` - Specify the port number of the database server.
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.

Datadirect Type 4 Driver for Sybase Database

The JAR file for the DataDirect driver is `sybase.jar`. Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: DataDirect-Sybase
- DataSource Classname: `com.ddtek.jdbcx.sybase.SybaseDataSource`
- Properties:
  - `serverName` - Specify the host name or IP address of the database server.
  - `portNumber` - Specify the port number of the database server.
  - `databaseName` - Set as appropriate. This is optional.
  - `user` - Set as appropriate.

- `password` - Set as appropriate.

> **NOTE**
>
> In some situations, using this driver can cause exceptions to be thrown because the driver creates a stored procedure for every parameterized PreparedStatement by default. If this situation arises, add the property `PrepareMethod`, setting its value to `direct`.

Inet Merlia Driver for Microsoft SQL Server Database

The JAR file for the Inet Microsoft SQL Server driver is `Merlia.jar`. Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: MicrosoftSqlServer
- DataSource Classname: `com.inet.tds.TdsDataSource`
- Properties:
  - `serverName` - Specify the host name or IP address and the port of the database server.
  - `portNumber` - Specify the port number of the database server.
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.

Inet Sybelux Driver for Sybase Database

The JAR file for the Inet Sybase driver is `Sybelux.jar`. Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: Sybase

- DataSource Classname: `com.inet.syb.SybDataSource`
- Properties:
  - `serverName` - Specify the host name or IP address of the database server.
  - `portNumber` - Specify the port number of the database server.
  - `databaseName` - Set as appropriate. Do not specify the complete URL, only the database name.
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.

Jconnect Type 4 Driver for Sybase ASE 12.5 Database

The JAR file for the Sybase driver is `jconn4.jar`. Configure the connection pool using the following settings:

- Name: Use this name when you configure the JDBC resource later.
- Resource Type: Specify the appropriate value.
- Database Vendor: Sybase
- DataSource Classname: Specify one of the following:

```
com.sybase.jdbc4.jdbc.SybDataSource
com.sybase.jdbc4.jdbc.SybXADataSource
```

- Properties:
  - `serverName` - Specify the host name or IP address of the database server.
  - `portNumber` - Specify the port number of the database server.
  - `databaseName` - Set as appropriate. Do not specify the complete URL, only the database name.
  - `user` - Set as appropriate.
  - `password` - Set as appropriate.

- `BE_AS_JDBC_COMPLIANT_AS_POSSIBLE` - Set to `true`.
- `FAKE_METADATA` - Set to `true`.

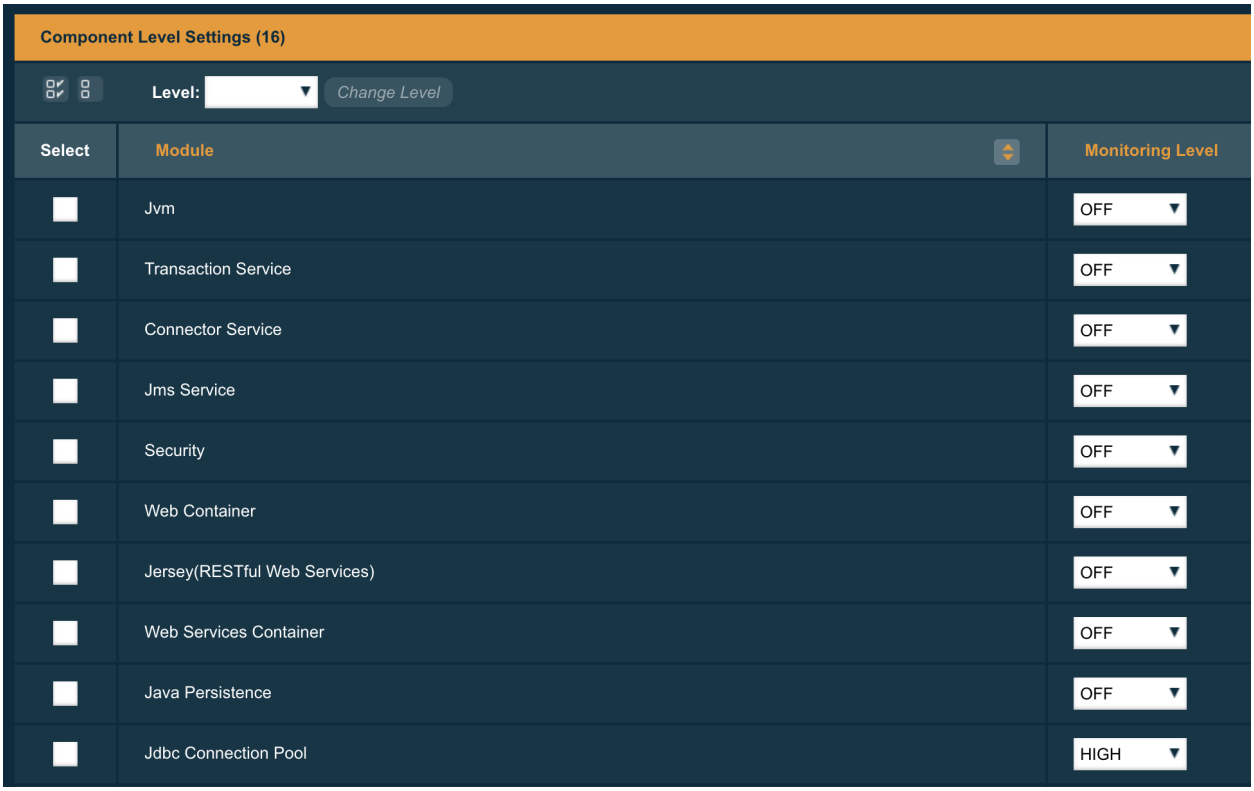
SQL Management in Payara Server

Payara Server keeps track of the recently executed SQL statements for monitoring purposes. This way, one can review which SQL statements are executed, how many times the same statement is executed and how long it took.

Activate Overview

The activation of the collection is controlled by the monitoring service.

When using the Web Administration console, go to *Configuration* → *[instance-configuration (like server-config)]* → *Monitoring:* and select the level *High* for the *JDBC Connection pool* component.



When using the *asadmin* CLI, you can execute the following command to set the *Level Value*

```
set configs.config.server-config.monitoring-service.module-monitoring-levels.jdbc-
```

```
connection-pool=HIGH
```

The information can be consulted via the JMX beans or the Rest monitoring interface. These need to be activated also, see jmx-monitoring-service and rest-monitoring on how to do this.

Viewing Statements

When you want to retrieve the information through the JMX channel, you can find the information in the following MBean:

```
amx:pp=/mon/server-mon[server],type=jdbcra-mon,name=resources/<pool-name>
```

When using the Rest monitoring interface, you can access the following URL

```
http://localhost:4848/monitoring/domain/server/resources/<pool-name>
```

In both cases, the *<pool-name>* is the actual pool name you have created within Payara Server and which you like to retrieve the information for.

The properties we want to highlight here are

`frequsedsqlqueries`: Contains the list of all SQL statements and their execution count, which are executed during the collection time frame. `slowSqlQueries`: Contains again the list of all SQL statements but this time with the maximum execution time they took.

Configuring Parameters

The SQL statement collection has a few configuration values which have default values.

- Timeframe: By default, the info about the SQL statements are retained during 5 minutes
- Cache size: The information is kept in a cache which has by default place for 100000 items.

These values can be configured by defining some key-value pairs in the properties for

the connection pool. This can be done during the creation or afterward by updating the pool.

- TIME-TO-KEEP-QUERIES-IN-MINUTES: This defines the number of minutes the SQL statement info is retained.
- MAX-CACHE-SIZE: This defines the maximum size of the cache.

TIP | Setting the maximum size of the cache to `0` will disable it completely.

When the cache size is too small, you see an information message in the log to indicate that not all statements could be retained.

Slow SQL Logger

Many performance problems in Enterprise Applications can be traced to slow database access. Payara Server includes capabilities to detect and log slow SQL queries executed via a JDBC Connection pool. The Slow SQL logger monitors all queries executed on the connection pool and if they exceed a configurable execution time in seconds a warning message is logged into the server log.

The warning message (see below) logs the SQL query and the stack trace to the code executing the query. This enables rapid diagnosis by pinpointing the exact lines of code to investigate.

Configuring the Slow SQL Logger

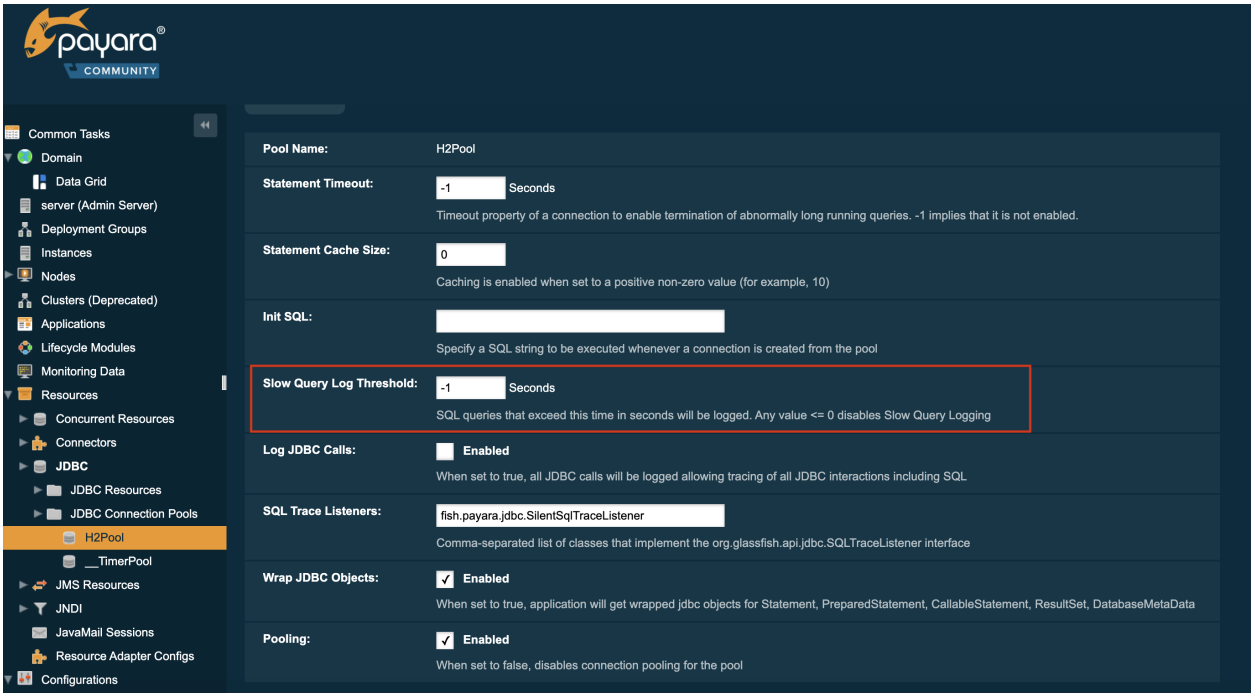
Ensure Wrap JDBC Objects is enabled in your JDBC Connection Pool (enabled by default) for the Slow SQL Logger to function correctly.

There are a number of ways to configure the slow SQL threshold time on a JDBC connection pool.

Using the Administration Console

To configure slow SQL logging on a JDBC connection pool via the administration console, navigate to the Connection Pool's **Advanced Properties** tab. Then specify the **Slow Query Log Threshold** time in seconds or use a decimal value to signify

milliseconds.



TIP | setting the threshold to `-1` will disable the feature

Using Asadmin Commands

The **Slow Query Log Threshold** time for a JDBC connection pool can also be configured with an `asadmin` command, by using the `set` command to set the `slow-query-threshold-in-seconds` property, like in the following example:

```
asadmin set domain.resources.jdbc-connection-pool.__TimerPool.slow-query-threshold-in-seconds=50
```

You can also set the threshold time to milliseconds simply by passing a decimal value. Below is an example of setting the threshold on the `__TimerPool` JDBC connection pool to **200** milliseconds:

```
asadmin set domain.resources.jdbc-connection-pool.__TimerPool.slow-query-threshold-in-seconds=0.2
```

Configuring SQL Trace Listeners

Payara Server provides support for custom SQL Trace Listeners. A **SQL Trace Listener** is

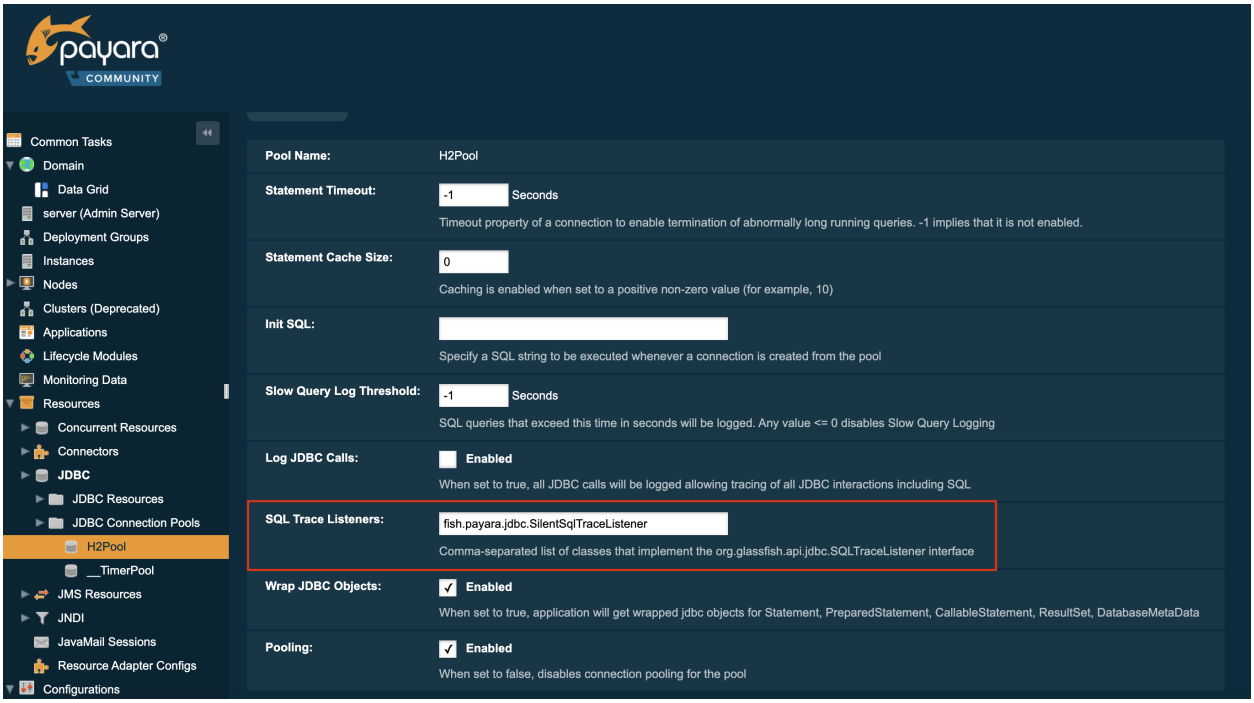
registered against a data source and is called after each method call made on the JDBC connection pool.

More information on how to implement a SQL Trace listener can be found in the sql-trace-listeners.

Using the Administration Console

SQL Trace Listeners can be enabled on a JDBC connection pool through the administration web console. Navigate to the **Advanced Tab** of your connection pool by selecting the JDBC JDBC Connection Pools **<Your Connection pool>**.

Proceed to add the fully qualified class name of your SQL Trace Listener implementation class in the **SQL Trace Listeners** field:



IMPORTANT	The implementation class must be on the server's *classpath* for it to work correctly.

Using Asadmin Commands

SQL Trace Listeners can be enabled on a JDBC connection pool using with an `asadmin` commands by using the `set` command to set the `fish.payara.lsql-trace-`

`listeners` to the fully qualified class name of your listener implementation, like on the following example:

```
asadmin set domain.resources.jdbc-connection-pool.__TimerPool.sql-trace-listeners
=fish.payara.examples.payaramicro.datasource.example.CustomSQLTracer
```

Administering EIS Connectivity

This chapter provides information and procedures for administering connections to enterprise information system (EIS) data in the Payara Server environment by using the `asadmin` command-line utility.

NOTE

If you installed the Web Profile, connector modules that use only outbound communication features and work management that does not involve inbound communication features are supported. Other connector features are supported only in the Full Platform Profile.

Instructions for accomplishing the tasks in this chapter by using the Administration Console are contained in the Administration Console online help.

For information about database connectivity, see "Administering Database Connectivity".

About EIS Connectivity

Enterprise information system (EIS) refers to any system that holds the data of an organization. It can be a mainframe, a messaging system, a database system, or an application. Connection resources are used by applications and modules to Create an Administered Object applications and modules to access EIS software.

The key elements of EIS connectivity are the following:

- **Connector Module**. A connector module also called a resource adapter, is a Jakarta EE component that enables applications to interact with EIS software. A connector module is used by Payara Server to implement Java Message Service (JMS). Like other Jakarta EE modules, a connector module is installed when it is deployed. For

instructions on creating a connector module, see "Developing Connectors" in the Payara Server Application Development section.

- **Connector Connection Pool**. A connector connection pool is a group of reusable connections for a particular EIS. A connector connection pool is created when you specify the connector module that is associated with the pool. For administration procedures, see administering-connector-connection-pools.
- **Connector Resource**. A connector resource is a program object that provides an application with a connection to an EIS. A connector resource is created when you specify its JNDI name and its associated connection pool. The JNDI name of a connector resource for an EIS is usually in the `java:comp/env/eis-specific` subcontext. For administration procedures, see administering-connector-resources.
- **Connector Module Configuration**. A connector module configuration is the information that resides in the domain configuration file (`domain.xml`) for the particular connector module (resource adapter). For administration procedures, see administering-the-resource-adapter-configuration.
- **Connector Security Map**. A connector security map associates the caller identity of the application (principal or user group) to a suitable EIS principal or group. For administration procedures, see administering-connector-security-maps.
- **Connector Work Security Map**. A connector work security map associates the caller identity of the work submitted by the connector module (resource adapter) EIS principal or EIS user group to a suitable principal or user group in the Payara Server security domain. For administration procedures, see administering-connector-work-security-maps.
- **Administered Object**. An administered object provides specialized functionality for an application, such as providing access to a parser that is specific to the connector module and its associated EIS. For administration procedures, see administering-administered-objects.

At runtime, the following sequence occurs when an application connects to an EIS:

1. The application gets the connector resource (data source) associated with the EIS by making a call through the JNDI API. Using the JNDI name of the connector resource, the naming and directory service locates the resource. Each EIS resource specifies a connector connection pool.
2. Using the connector resource, the application gets an EIS connection. Payara Server retrieves a physical connection from the connection pool that corresponds to the EIS resource. The pool defines connection attributes such as the EIS name, username, and password.
3. After the EIS connection is established, the application can read, modify, and add data to the EIS. The application accesses the EIS information by making calls to the JMS API.
4. When the application is finished accessing the EIS, the application closes the connection and returns the connection to the connection pool.

TIP

You can view the full syntax and options of the subcommand by typing `asadmin help `subcommand` at the command line. For Example:

```
asadmin help create-connector-connection-pool
```

Administering Connector Connection Pools

After a connector module has been deployed, you are ready to create a connector connection pool for it.

To Create a Connector Connection Pool

Use the `create-connector-connection-pool` subcommand in remote mode to create a connector connection pool for a deployed connector module. When you are building the connector connection pool, certain data specific to the EIS will be required. The value in the mandatory `--connectiondefintion` option provides the EIS info.

Multiple connector resources can specify a single connection pool.

Creating a connector connection pool is a dynamic event and does not require server

restart. However, there are some parameters that do require server restart. See configuration-changes-that-require-restart.

Before creating the connector connection pool, the connector must be installed.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create the connector connection pool by using the "`create-connector-connection-pool`" subcommand. Information about properties for the subcommand is included in this help page.
3. If needed, restart the server. Some properties require server restart. See configuration-changes-that-require-restart. If your server needs to be restarted, see to-restart-a-domain.
4. You can verify that a connection pool is usable by using the `ping-connection-pool` subcommand. For instructions, see to-contact-ping-a-connection-pool.

Example 12-1 Creating a Connector Connection Pool

```
asadmin> create-connector-connection-pool --steadypoolsize 20 --maxpoolsize 100
--poolresize 2 --maxwait 60000 --raname jmsra --connectiondefinition
jakarta.jms.QueueConnectionFactory jms/qConnPool

Command create-connector-connection-pool executed successfully
```

To List Connector Connection Pools

Use the `list-connector-connection-pools` subcommand in remote mode to list the pools that have been created.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the connector connection pools by using the "`list-connector-connection-pools`)" subcommand.

Example 12-2 Listing Connector Connection Pools

```
asadmin> list-connector-connection-pools
jms/qConnPool
Command list-connector-connection-pools executed successfully
```

To Connect to (Ping) or Reset (Flush) a Connector Connection Pool

Use the `ping-connection-pool` or `flush-connection-pool` subcommands in remote mode to perform these tasks on a connection pools. See to-contact-ping-a-connection-pool or to-reset-flush-a-connection-pool for instructions.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Connect to or reset a connector connection pool by using the "`flush-connection-pool`" subcommand or the "`ping-connection-pool`" subcommand.

To Update a Connector Connection Pool

Use the `get` and `set` subcommands to view and change the values of the connector connection pool properties.

1. List the connector connection pools by using the "`list-connector-connection-pools`" subcommand.
2. View the properties of the connector connection pool by using the "`get`" subcommand.

   For example:

   ```
   asadmin> get domain.resources.connector-connection-pool.*
   ```

3. Set the property of the connector connection pool by using the "`set`" subcommand.

   For example:

   ```
   asadmin> set domain.resources.connector-connection-pool
   .conectionpoolname.validate-atmost-once-period-in-seconds=3
   ```

4. If needed, restart the server. Some properties require server restart. See configuration-changes-that-require-restart. If your server needs to be restarted, see to-restart-a-domain.

To Delete a Connector Connection Pool

Use the `delete-connector-connection-pool` subcommand in remote mode to

remove a connector connection pool.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the connector connection pools by using the "`list-connector-connection-pools`" subcommand.
3. If necessary, notify users that the connector connection pool is being deleted.
4. Delete the connector connection pool by using the "`delete-connector-connection-pool`" subcommand.

Example 12-3 Deleting a Connector Connection Pool

```
asadmin> delete-connector-connection-pool --cascade=false jms/qConnPool
Command delete-connector-connection-pool executed successfully
```

Administering Connector Resources

A connector resource provides an application or module with the means of connecting to an EIS. Typically, you create a connector resource for each EIS that is accessed by the applications deployed in the domain.

To Create a Connector Resource

Use the `create-connector-resource` subcommand in remote mode to register a new connector resource with its JNDI name.

Creating a connector resource is a dynamic event and does not require server restart. However, there are some parameters that do require server restart. See configuration-changes-that-require-restart.

Before creating a connector resource, you must first create a connector connection pool. For instructions, see to-create-a-connector-connection-pool.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create the connector resource by using the "`create-connector-resource`" subcommand. Information about properties for the subcommand is included in this help page.

3. If needed, restart the server. Some properties require server restart. See configuration-changes-that-require-restart. If your server needs to be restarted, see to-restart-a-domain.

Example 12-4 Creating a Connector Resource

```
asadmin> create-connector-resource --poolname jms/qConnPool
--description "creating sample connector resource" jms/qConnFactory
Command create-connector-resource executed successfully
```

To List Connector Resources

Use the `list-connector-resources` subcommand in remote mode to list the connector resources that have been created.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the connector connection pools by using the "`list-connector-resources`" subcommand.

Example 12-5 Listing Connector Resources

```
asadmin> list-connector-resources
jms/qConnFactory
Command list-connector-resources executed successfully
```

To Update a Connector Resource

Use the `get` and `set` subcommands to view and change the values of the connector resource properties.

1. List the connector connection pools by using the "`list-connector-resources`" subcommand.
2. View the properties of the connector resource by using the "`get`" subcommand.

   For example

   ```
   asadmin> get domain.resources.connector-resource.jms/qConnFactory
   ```

3. Set the property of the connector resource by using the "`set`" subcommand.

For example:

```
asadmin> set domain.resources.connector-resource.jms/qConnFactory.enabled=true
```

4. If needed, restart the server. Some properties require server restart. See configuration-changes-that-require-restart. If your server needs to be restarted, see to-restart-a-domain.

To Delete a Connector Resource

Use the `delete-connector-resource` subcommand in remote mode to remove a connector resource by specifying the JNDI name.

Before deleting a resource, all associations with the resource must be removed.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the connector connection pools by using the "`list-connector-resources`" subcommand.
3. If necessary, notify users that the connector resource is being deleted.
4. Delete the connector resource by using the "`delete-connector-resource`" subcommand.

Example 12-6 Deleting a Connector Resource

```
asadmin> delete-connector-resource jms/qConnFactory
Command delete-connector-resources executed successfully
```

Administering the Resource Adapter Configuration

To Create Configuration Information for a Resource Adapter

Use the `create-resource-adapter-config` subcommand in remote mode to create configuration information for a resource adapter, also known as a connector module. You can run the subcommand before deploying a resource adapter, so that the configuration information is available at the time of deployment. The resource adapter configuration can also be created after the resource adapter is deployed. In this situation, the resource adapter is restarted with the new configuration.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create configuration information by using the "`create-resource-adapter-config`" subcommand. Information about properties for the subcommand is included in this help page.

Example 12-7 Creating a Resource Adapter Configuration

```
asadmin> create-resource-adapter-config --property foo=bar
--threadpoolid mycustomerthreadpool ra1
Command create-resource-adapter-config executed successfully
```

To List Resource Adapter Configurations

Use the `list-resource-adapter-configs` subcommand in remote mode to list the configuration information contained in the domain configuration file (`domain.xml`) for the specified resource adapter (connector module).

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the configurations for a resource adapter by using the "`list-resource-adapter-configs`" subcommand.

Example 12-8 Listing Configurations for a Resource Adapter

```
asadmin> list-resource-adapter-configs
ra1
ra2
Command list-resource-adapter-configs executed successfully
```

To Update a Resource Adapter Configuration

Use the `get` and `set` subcommands to view and change the values of the resource adapter configuration properties.

1. List the configurations for a resource adapter by using the "`list-resource-adapter-configs`" subcommand.
2. View the properties of the connector resource by using the "`get`" subcommand.

For example:

```
asadmin>get domain.resources.resource-adapter-config.ra1.*
```

3. Set the property of the connector resource by using the "`set`" subcommand.

   For example:

```
asadmin> set domain.resources.resource-adapter-config.ra1.raSpecificProperty
=value
```

To Delete a Resource Adapter Configuration

Use the `delete-resource-adapter-config` subcommand in remote mode to delete the configuration information contained in the domain configuration file (`domain.xml`) for a specified resource adapter (connector module).

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the configurations for a resource adapter by using the "`list-resource-adapter-configs`" subcommand.
3. Delete the configuration for a resource adapter by using the "`delete-resource-adapter-config`" subcommand.

   Example 12-9 Deleting a Resource Adapter Configuration

```
asadmin> delete-resource-adapter-config ra1
Command delete-resource-adapter-config executed successfully
```

Administering Connector Security Maps

The EIS is any system that holds the data of an organization. It can be a mainframe, a messaging system, a database system, or an application. The connector security map is used to map the application's credentials to the EIS credentials.

A security map applies to a particular connector connection pool. One or more named security maps can be associated with a connector connection pool.

To Create a Connector Security Map

Use the `create-connector-security-map` subcommand in remote mode to create a security map for the specified connector connection pool. If the security map is not present, a new one is created. You can specify back-end EIS principals or back-end EIS user groups. The connector security map configuration supports the use of the wild card asterisk `(*)` to indicate all users or all user groups.

You can also use this subcommand to map the caller identity of the application (principal or user group) to a suitable EIS principal in container-managed authentication scenarios.

For this subcommand to succeed, you must have first created a connector connection pool. For instructions, see to-create-a-connector-connection-pool.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create a connector security map by using the "`create-connector-security-map`" subcommand. Information about the options for the subcommand is included in this help page.
3. If needed, restart the server. Some properties require server restart. See configuration-changes-that-require-restart. If your server needs to be restarted, see to-restart-a-domain.

Example 12-10 Creating a Connector Security Map

```
asadmin> create-connector-security-map --poolname connector-pool1
--principals principal1, principal2 --mappedusername backend-username securityMap1
Command create-connector-security-map executed successfully
```

To List Connector Security Maps

Use the `list-connector-security-maps` subcommand in remote mode to list the existing security maps belonging to the specified connector connection pool. You can get a simple listing of the connector security maps for a connector connection pool, or you can get a more comprehensive listing that shows the principals of the map.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List existing connector connection pools by using the "`list-connector-connection-pools`" subcommand.
3. List the security maps for a specific connector connection pool by using the "`list-connector-security-maps`" subcommand.

Example 12-11 Listing All Connector Security Maps for a Connector Connection Pool

```
asadmin> list-connector-security-maps connector-Pool1
securityMap1
Command list-connector-security-maps executed successfully.
```

Example 12-12 Listing Principals for a Specific Security Map for a Connector Connection Pool

```
asadmin> list-connector-security-maps --securitymap securityMap1 connector-Pool1
principal1
principal1
Command list-connector-security-maps executed successfully.
```

Example 12-13 Listing Principals of All Connector Security Maps for a Connector Connection Pool

```
asadmin> list-connector-security-maps --verbose connector-Pool1
securityMap1
principal1
principal1
Command list-connector-security-maps executed successfully.
```

To Update a Connector Security Map

Use the `update-connector-security-map` subcommand in remote mode to create or modify a security map for the specified connector connection pool.

1. Ensure that the server is running.

   Remote subcommands require a running server.
2. List existing connector security maps by using the "`list-connector-security-maps`" subcommand.

3. Modify a security map for a specific connector connection pool by using the "`update-connector-security-map`" subcommand.
4. If needed, restart the server. Some properties require server restart. See configuration-changes-that-require-restart. If your server needs to be restarted, see to-restart-a-domain.

Example 12-14 Updating a Connector Security Map

```
asadmin> update-connector-security-map --poolname connector-pool1
--addprincipals principal1, principal2 securityMap1
Command update-connector-security-map executed successfully.
```

To Delete a Connector Security Map

Use the `delete-connector-security-map` subcommand in remote mode to delete a security map for the specified connector connection pool.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List existing connector connection pools by using the "`list-connector-connection-pools`" subcommand.
3. Delete a security map for a specific connector connection pool by "`delete-connector-security-map`" subcommand. Information about options for this subcommand is included in this help page.

   Example 12-15 Deleting a Connector Security Map

   ```
   asadmin> delete-connector-security-map --poolname connector-pool1 securityMap1

   Command delete-connector-security-map executed successfully
   ```

Administering Connector Work Security Maps

The EIS is any system that holds the data of an organization. It can be a mainframe, a messaging system, a database system, or an application. The connector work security map is used to is used to map the EIS credentials to the credentials of Payara Server security domain.

A security map applies to a particular connector connection pool. One or more named

security maps can be associated with a connector connection pool.

To Create a Connector Work Security Map

Use the `create-connector-work-security-map` subcommand in remote mode to map the caller identity of the work submitted by the connector module (resource adapter) EIS principal or EIS user group to a suitable principal or user group in the Payara Server security domain. One or more work security maps can be associated with a connector module.

The connector security map configuration supports the use of the wild card asterisk (*) to indicate all users or all user groups.

Before creating a connector work security map, you must first create a connector connection pool. For instructions, see to-create-a-connector-connection-pool.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create the connector work security map by using the "`create-connector-work-security-map`" subcommand. Information about properties for the subcommand is included in this help page.
3. If needed, restart the server. Some properties require server restart. See configuration-changes-that-require-restart. If your server needs to be restarted, see to-restart-a-domain.

Example 12-16 Creating Connector Work Security Maps

```
asadmin> create-connector-work-security-map --raname my-resource-adapter-name
--principalsmap eis-principal-1=server-principal-1,eis-principal-2=server-
principal-2,
eis-principal-3=server-principal-1 workSecurityMap1

asadmin> create-connector-work-security-map --raname my-resource-adapter-name
--groupsmap eis-group-1=server-group-1,eis-group-2=server-group-2,
eis-group-3=server-group-1 workSecurityMap2
Command create-connector-work-security-map executed successfully
```

+ The following examples create `workSecurityMap1` and `workSecurityMap2` for `my-resource-adapter-name`.

To List Connector Work Security Maps

Use the `list-connector-work-security-maps` subcommand in remote mode to list the work security maps that belong to a specific connector module.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the connector work security maps by using the "`list-connector-work-security-maps`" subcommand.

Example 12-17 Listing the Connector Work Security Maps

```
asadmin> list-connector-work-security-maps generic-ra
generic-ra-groups-map: EIS group=eis-group, mapped group=payara-group
generic-ra-principals-map: EIS principal=eis-bar, mapped principal=bar
generic-ra-principals-map: EIS principal=eis-foo, mapped principal=foo
Command list-connector-work-security-maps executed successfully.
```

To Update a Connector Work Security Map

Use the `update-connector-work-security-map` subcommand in remote to modify a work security map that belongs to a specific resource adapter (connector module).

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the connector work security maps by using the "`list-connector-work-security-maps`" subcommand.
3. If necessary, notify users that the connector work security map is being modified.
4. Update a connector work security map by using the "`update-connector-work-security-map`" subcommand.

Example 12-18 Updating a Connector Work Security Map

```
asadmin> update-connector-work-security-map --raname generic-ra
--removeprincipals eis-foo generic-ra-principals-map
Command update-connector-work-security-map executed successfully.
```

To Delete a Connector Work Security Map

Use the `delete-connector-work-security-map` subcommand in remote mode to

delete a work security map that belongs to a specific connector module (resource adapter).

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the connector work security maps by using the "`list-connector-work-security-maps`" subcommand.
3. Delete a connector work security map by using the "`delete-connector-work-security-map`" subcommand.

Example 12-19 Deleting a Connector Work Security Map

```
asadmin> delete-connector-work-security-map --raname my_ra worksecuritymap1
Command delete-connector-work-security-map executed successfully.
```

Administering Administered Objects

Packaged within a connector module, an administered object provides specialized functionality for an application. For example, an administered object might provide access to a parser that is specific to the connector module and its associated EIS.

To Create an Administered Object

Use the `create-admin-object` subcommand to create an administered object resource. When creating an administered object resource, name-value pairs are created, and the object is associated to a JNDI name.

Before You Begin The resource adapter must be deployed before running this subcommand (`jmsrar.rar`).

1. Create an administered object by using the "`create-admin-object`" subcommand. Information about properties for the subcommand is included in this help page.
2. If needed, restart the server. Some properties require server restart. See configuration-changes-that-require-restart. If your server needs to be restarted, see to-restart-a-domain.

Example 12-20 Creating an Administered Object

```
asadmin> create-admin-object --restype jakarta.jms.Queue --raname jmsra
--description "sample administered object" --property Name=sample_jmsqueue
jms/samplequeue
Command create-admin-object executed successfully
```

To List Administered Objects

Use the `list-admin-object` subcommand in remote mode to list the existing administered objects.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the administered objects by using the "`list-admin-objects`" subcommand.

Example 12-21 Listing Administered Objects

```
asadmin> list-admin-objects
jms/samplequeue
Command list-admin-objects executed successfully
```

To Update an Administered Object

Use the `get` and `set` subcommands to view and change the values of the administered objects properties.

1. List the administered objects by using the "`list-admin-objects`" subcommand.
2. View the properties of the administered object by using the "`get`" subcommand.

   For example:

   ```
   asadmin> get domain.resources.admin-object-resource.jms/samplequeue.*
   ```

3. Set the property of the administered object by using the "`set`" subcommand.

   For example:

   ```
   asadmin> set domain.resources.admin-object-resource.jms/samplequeue.enabled
   =false
   ```

4. If needed, restart the server. Some properties require server restart. See see

configuration-changes-that-require-restart. If your server needs to be restarted, see see to-restart-a-domain.

To Delete an Administered Object

Use the `delete-admin-object` subcommand to delete an administered objects.

1. List the administered objects by using the "`list-admin-objects`" subcommand.
2. If necessary, notify users that the administered object is being deleted.
3. Delete an administered object by using the "`delete-admin-object`" subcommand.

Example 12-22 Deleting an Administered Object

```
asadmin> delete-admin-object jms/samplequeue
Command delete-admin-object executed successfully
```

Administering HTTP Connectivity

This chapter provides procedures for performing HTTP connectivity tasks in the Payara Server environment by using the `asadmin` command-line utility.

Instructions for accomplishing the tasks in this chapter by using the Administration Console are contained in the Administration Console online help.

About Internet Connectivity

The HTTP service provides functionality for deploying web applications and for making deployed web applications accessible by Internet clients, either in a single application server instance or in a cluster of multiple server instances. HTTP services are provided by two kinds of related objects: listeners and virtual servers.

For more information about clusters, see the "Payara Server High Availability section".

About HTTP Network Listeners

An HTTP listener, also known as a network listener, is a listen socket that has an Internet Protocol (IP) address, a port number, a server name, and a default virtual server. Each virtual server provides connections between the server and clients through one or

more listeners. Each listener must have a unique combination of port number and IP address. For example, an HTTP listener can listen for a host on all configured IP addresses on a given port by specifying the IP address `0.0.0.0`.

A listener can listen on a specific port on all network interfaces by using the IP `0.0.0.0`, and a configured port. Alternatively, the listener can specify a unique IP address for each listener while using the same port.

A virtual server will then be used to select a combination of network listeners to use for an application. For example, Payara Server has two default network listeners for user deployed applications: an insecure listener on port 8080 (`http-listener-1`), and a secure listener on port 8181 (`http-listener-2`).

Because an HTTP listener is a combination of IP address and port number, you can have multiple HTTP listeners with the same IP address and different port numbers, or with different IP addresses and the same port number (if your host was configured to respond to these addresses). However, if an HTTP listener uses the `0.0.0.0` IP address, which listens on all IP addresses on a port, you cannot create HTTP listeners for additional IP addresses that listen on the same port for a specific IP address. For example, if an HTTP listener uses `0.0.0.0:8080` (all IP addresses on port 8080), another HTTP listener cannot use `1.2.3.4:8080`.

The host running the Payara Server typically has access to only one IP address. HTTP listeners typically use the `0.0.0.0` IP address and different port numbers, with each port number serving a different purpose. However, if the host does have access to more than one IP address, each address can serve a different purpose.

To access a web application deployed on Payara Server, use the URL `http://localhost:8080/` (or `https://localhost:8081/` for a secure application), along with the context root specified for the web application.

To access the Administration Console, use the URL `https://localhost:4848/` or `http://localhost:4848/asadmin/` (console default context root).

About Virtual Servers

A virtual server, sometimes called a virtual host, is an object that allows the same physical server to host multiple Internet domain names.

All virtual servers hosted on the same physical server share the IP address of that physical server. A virtual server associates a domain name for a server (such as `www.aaa.com`) with the particular server on which Payara Server is running. Each virtual server must be registered with the DNS server for your network.

NOTE	Do not confuse an Internet domain with the administrative domain of Payara Server.

For example, assume that you want to host the following domains on your physical server: `www.aaa.com`, `www.bbb.com`, and `www.ccc.com`. Assume that these domains are respectively associated with web modules `web1`, `web2`, and `web3`. This means that the following URLs are handled by your physical server:

```
http://www.aaa.com:8080/web1
http://www.bbb.com:8080/web2
http://www.ccc.com:8080/web3
```

The first URL is mapped to virtual server `www.aaa.com`, the second URL is mapped to virtual server `www.bbb.com`, and the third is mapped to virtual server `www.ccc.com`. For this mapping to work, `www.aaa.com`, `www.bbb.com`, and `www.ccc.com` must all resolve to your physical server's IP address and each virtual server must be registered with the DNS server for your network. In addition, on a UNIX system, add these domains to your `/etc/hosts` file (if the setting for `hosts` in your `/etc/nsswitch.conf` file includes `files`).

Administering HTTP Network Listeners

By default, when Payara Server starts, the following HTTP listeners are started automatically:

- HTTP listeners associated with the virtual server named `server`:

- The listener named `http-listener-1` does not have security enabled.
- The listener named `http-listener-2` has security enabled

- An HTTP listener named `admin-listener`, associated with the virtual server named `__asadmin`. For this listener, security is not enabled.

The following table describes the Payara Server default ports for the listeners that use ports.

Table 4. Default Ports for Listeners

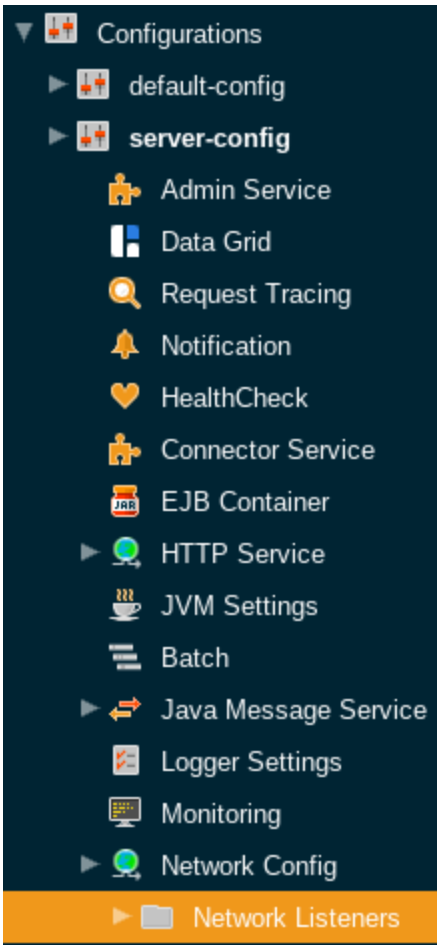
Listener	Default Port	Description
Administrative server	`4848`	A domain's administrative server is accessed by the Administration Console and the `asadmin` utility.
HTTP	`8080`	The web server listens for HTTP requests on a port. To access deployed web applications and services, clients connect to this port.
HTTPS	`8181`	Web applications configured for secure communications listen on a separate port.

Network Listeners Configuration

NOTE

When configuring network listeners from the admin console, there are some visible configuration options that transparently configure the protocol being used.
If a configuration property isn't shown here but is visible in the admin console, see Protocols.

Existing network listeners can be configured from the admin console or with asadmin commands. The network listener configurations can be found in the admin console under **Configurations** ▯ **your-config** ▯ **Network Config** ▯ **Network Listeners**.



The following fields can be configured for network listeners:

Table 5. Network Listeners Configuration

Configuration settings	Description	Asadmin Command
Name	The unique name of the network listener. This cannot be edited from the admin console.	`set configs.config.server-config.network-config.network-listeners.network-listener.${network-listener-name}.name=value`
Protocol	The protocol used by the network listener. This cannot be edited from the admin console.	`set configs.config.server-config.network-config.network-listeners.network-listener.${network-listener-name}.protocol=${protocol-name}`

Configuration settings	Description	Asadmin Command
Status	The status of the network listener. This determines whether the listener will be actively listening for requests.	`set configs.config.server-config.network-config.network-listeners.network-listener.${network-listener-name}.enabled=[true/false]`
JK Enabled	Whether mod_jk support should be enabled for this listener. Should only be enabled if mod_jk is being explicitly configured on an Apache Web Server in conjunction with Payara Server.	`set configs.config.server-config.network-config.network-listeners.network-listener.${network-listener-name}.jk-enabled=[true/false]`
JK Configuration File	The location of the mod_jk configuration file. Defaults to ${com.sun.aas.instanceRoot}/config/glassfish-jk.properties.	`set configs.config.server-config.network-config.network-listeners.network-listener.${network-listener-name}.jk-configuration-file=/location/to/file`
Port	The port used by the network listener. The selected port will be used to listen for requests.	`set configs.config.server-config.network-config.network-listeners.network-listener.${network-listener-name}.port=value`
Address	The address used by the network listener. 0.0.0.0 corresponds to all addresses.	`set configs.config.server-config.network-config.network-listeners.network-listener.${network-listener-name}.address=value`
Transport	The transport used by the network listener. This cannot be edited from the admin console. Defaults to tcp.	`set configs.config.server-config.network-config.network-listeners.network-listener.${network-listener-name}.transport=${transport-name}`
Thread Pool	The thread pool used by the network listener. This determines the pool of threads used to service requests.	`set configs.config.server-config.network-config.network-listeners.network-listener.${network-listener-name}.thread-pool=${thread-pool-name}`

To Create an Internet Connection

Use the subcommands in this procedure to create an internet connection with the full

range of listener options. A network listener is created behind the scenes. For the shortcut version of this process , see to-create-an-http-network-listener.

1. Ensure that the server is running.

   Remote subcommands require a running server.

2. Create an HTTP or HTTPS protocol by using the "`create-protocol`" subcommand with the `--securityenabled` option.

   To use the built-in `http-listener-1` HTTP protocol, or `http-listener-2` HTTPS protocol, skip this step.

3. Create an HTTP configuration by using the "`create-http`" subcommand.

   To use a built-in protocol, skip this step.

4. Create a transport by using the "`create-transport`" subcommand.

   To use the built-in `tcp` transport, skip this step.

5. Create a thread pool by using the "`create-threadpool`" subcommand.

   To avoid using a thread pool, or to use the built-in `http-thread-pool` thread pool, skip this step.

   For additional thread pool information, see administering-thread-pools.

6. Create an HTTP listener by using the "`create-network-listener`" subcommand.

   Specify a protocol and transport, optionally a thread pool.

7. To apply your changes, restart Payara Server.

   See to-restart-a-domain.

To Create an HTTP Network Listener

Use the `create-http-listener` subcommand or the `create-network-listener`

subcommand in remote mode to create a listener. These subcommands provide backward compatibility and also provide a shortcut for creating network listeners that use the HTTP protocol.

Behind the scenes, a network listener is created as well as its associated protocol, transport, and HTTP configuration. This method is a convenient shortcut, but it gives access to only a limited number of options. If you want to specify the full range of listener options, follow the instructions in to-create-an-internet-connection.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create an HTTP network listener by using the "`create-network-listener`" subcommand or the "`create-http-listener`" subcommand.
3. If needed, restart the server. If you edit the special HTTP network listener named `admin-listener`, you must restart the server for changes to take effect.

   See to-restart-a-domain.

Example - Creating an HTTP Listener

This example creates an HTTP listener named `sampleListener` that uses a non-default number of acceptor threads. Security is not enabled at runtime.

```
asadmin create-http-listener --listeneraddress `0.0.0.0` --listenerport 7272
--default-virtual-server server --servername host1.sun.com --acceptorthreads 100
--securityenabled=false --enabled=false sampleListener
Command create-http-listener executed successfully.
```

Example - Creating a Network Listener

This example a network listener named `sampleListener` that is not enabled at runtime:

```
asadmin create-network-listener --listenerport 7272 protocol http-1 --enabled=false
sampleListener
Command create-network-listener executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-http-listener` or `asadmin help create-network-listener`

at the command line.

To List HTTP Network Listeners

Use the `list-http-listeners` subcommand or the `list-network-listeners` subcommand in remote mode to list the existing HTTP listeners.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List HTTP listeners by using the "`list-http-listeners`" or "`list-network-listeners`" subcommand.

Example - Listing HTTP Listeners*

This example lists the HTTP listeners. The same output is given if you use the `list-network-listeners` subcommand.

```
asadmin list-http-listeners
admin-listener
http-listener-2
http-listener-1
Command list-http-listeners executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-http-listeners` or `asadmin help list-network-listeners` at the command line.

To Update an HTTP Network Listener

1. List HTTP listeners by using the "`list-http-listeners`" or "`list-network-listeners`" subcommand.
2. Modify the values for the specified listener by using the "`set`" subcommand.

   The listener is identified by its dotted name.

Example - Updating an HTTP Network Listener

This example changes `security-enabled` to `false` on `http-listener-2`.

```
asadmin set server.network-config.protocols.protocol.http-listener-2.security-
enabled=false
```

```
Command set executed successfully.
```

To Restart an HTTP Network Listener

The `restart-http-listeners` commands restarts all HTTP listeners but the `admin-listener` of the targeted instance(s). It is either used with the `--all` flag to target all instances or with a particular target instance, deployment group or cluster following the `--target` parameter.

When `target` and `all` parameters are both omitted the default target is the DAS.

NOTE This command only restarts all HTTP listeners except the `admin-listener`. IIOP listeners are not affected and may require a server restart to make changes take effect.

Example - Restarting an HTTP Network Listener

This example restarts the instance named `FlyingFish`:

```
asadmin restart-http-listeners --target FlyingFish
Command restart-http-listeners executed successfully.
```

To Delete an HTTP Network Listener

Use the `delete-http-listener` subcommand or the `delete-network-listener` subcommand in remote mode to delete an existing HTTP listener. This disables secure communications for the listener.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List HTTP listeners by using the "`list-http-listeners`" subcommand.
3. Delete an HTTP listener by using the "`delete-http-listener`" or "`delete-network-listener`" subcommand.
4. To apply your changes, restart Payara Server. See to-restart-a-domain.

Example - Deleting an HTTP Listener

This example deletes the HTTP listener named `sampleListener`:

```
asadmin delete-http-listener sampleListener
Command delete-http-listener executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-http-listener` or `asadmin help delete-network-listener` at the command line.

Configure an HTTP Listener for SSL

Use the `create-ssl` subcommand in remote mode to create and configure an SSL element in the specified listener. This enables secure communication for the listener.

1. Ensure that the server is running.

   Remote subcommands require a running server.

2. Configure an HTTP listener by using the "`create-ssl`" subcommand.
3. To apply your changes, restart Payara Server.

   See to-restart-a-domain.

Example - Configuring an HTTP Listener for SSL

This example enables the HTTP listener named `http-listener-1` for SSL:

```
asadmin create-ssl --type http-listener --certname sampleCert http-listener-1
Command create-ssl executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-ssl` at the command line.

Configure SNI for HTTP Listeners

Server Name Indication, SNI for short, allows you to use multiple SSL certificates with the same IP address. When enabled, the server will look for a certificate in the configured keystore using a nickname that matches the host name requested.

SNI can be enabled by configuring a network listener. Existing network listeners can be configured from the admin console or with asadmin commands. The network listener

configurations can be found in the admin console under **Configurations** → **<your-config>** → **Network Config** → **Network Listeners**.

From here, click on the SSL tab, and make sure that the **SNI Support** checkbox is ticked.

IMPORTANT	For SNI to take effect, the network listener must have security enabled by providing a default keypair and certificate, and by checking the **Security** box on the **HTTP** tab.

Configure Optional Client Authentication for SSL

In Payara Server, you can configure the SSL protocol of an HTTP listener such that it requests a certificate before permitting a client connection, but does not refuse a connection if the client does not provide one. To enable this feature, set the `client-auth` property of the SSL protocol to the value `want`. For example:

```
asadmin set configs.config.config-name.network-config.protocols.protocol.listener-name.ssl.client-auth=want
```

Configure a Custom SSL Implementation

In Payara Server, you can configure the SSL protocol an HTTP listener such that it uses a custom implementation of SSL. To enable this feature, set the `classname` property of the SSL protocol to the name of a class that implements the `com.sun.grizzly.util.net.SSLImplementation` interface. For example:

```
asadmin> set configs.config.config-name.network-config.protocols.protocol.listener-name.ssl.classname=SSLImplementation-class-name
```

By default, Payara Server uses the implementation `com.sun.enterprise.security.ssl.GlassfishSSLImpl` for the SSL protocol.

Delete SSL from an HTTP Listener

Use the `delete-ssl` subcommand in remote mode to delete the SSL element in the specified listener. This disables secure communications for the listener.

1. Ensure that the server is running.

   Remote subcommands require a running server.

2. Delete SSL from an HTTP listener by using the "`delete-ssl`" subcommand.
3. To apply your changes, restart Payara Server.

   See to-restart-a-domain.

Example - Deleting SSL from an HTTP Listener

This example disables SSL for the HTTP listener named `http-listener-1`:

```
asadmin delete-ssl --type http-listener http-listener-1
Command delete-ssl executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-ssl` at the command line.

Assign a Default Virtual Server to an HTTP Listener

1. In the Administration Console, open the HTTP Service component under the relevant configuration.
2. Open the HTTP Listeners component under the HTTP Service component.
3. Select or create a new HTTP listener.
4. Select from the Default Virtual Server drop-down list. For more information, see:

   to-assign-a-default-web-module-to-a-virtual-server.

For details, click the Help button in the Administration Console from the HTTP Listeners page.

Administering HTTP Protocols

A protocol is used by a network listener to determine the nature of the communication expected by the listener. For example, a protocol determines the timeout of a request, as well as whether HTTPS should be used. The reason the protocol is separate to the

listener is so that the same protocol configuration can be used to listen on multiple ports.

Existing protocols can be configured from the admin console or with asadmin commands. The protocol configurations can be found in the Admin Console under **Configurations your-config Network Config Protocols**.

Each HTTP listener has an HTTP protocol, which is created either by using the `create-protocol` subcommand or by using the built-in protocols that are applied when you follow the instructions in to-create-an-http-network-listener.

Protocol Configurations

Table 6. Security

Configuration settings	Description	Asadmin Command
Security	Whether security is enabled for the protocol. When disabled, all the SSL/TLS options are ignored.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.security-enabled=[true|false]`

Table 7. HTTP Settings

Configuration settings	Description	Asadmin Command
Server Name	The name of the server. This will be sent as the `Server` header in responses.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.server-name-enabled=value`
Default Virtual Server	The default virtual server used for the protocol. Establishes the request priority when multiple virtual servers are configured.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.default-virtual-server=value`
Redirect Port	The port used for SSL transport redirection.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.redirect-port=value`

Configuration settings	Description	Asadmin Command
Max Connections	The maximum number of requests per connection to keep in keep-alive mode.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.max-connections=value`
Timeout	The amount of time in seconds before a connection is deemed as idle. After this, the connection will no longer be in keep-alive state.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.timeout-seconds=value`
Upload Timeout Enabled	Enables closing a connection that is reading bytes slowly after the `configuration-http-upload-timeout-millis` setting is reached.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.upload-timeout-enabled=value`
Connection Upload Timeout	The timeout, in milliseconds, for uploads. **TIP** To disable this timeout, set the value to `-1`.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.connection-upload-timeout-millis=value`
Request Timeout	The timeout, in seconds, for requests. **TIP** To disable this timeout, set the value to `-1`.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.request-timeout-seconds=value`
Send Buffer Size	The size, in bytes, of the send buffer.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.send-buffer-size-bytes=value`
Header Buffer Length	The size, in bytes, of the buffer used to read the request data.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.header-buffer-length-bytes=value`

Configuration settings	Description	Asadmin Command
Max Post Size	The maximum size, in bytes, of POST actions.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.max-post-size-bytes=value`
Max Form Post Size	The maximum size, in bytes, of a POST form.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.max-form-post-size-bytes=value`
Max Save Post Size	The maximum size, in bytes, of a POST request that will be saved during authentication.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.max-save-post-size-bytes=value`
URI Encoding	The character set used to decode the request URIs received.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.uri-encoding=value`
Compression	Whether to enable HTTP/1.1 GZIP compression to save server bandwidth. Will be ignored when HTTP/2 is used.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.compression=[off/on/force]`
Compression Level	The level of compression to be used: • `-1` corresponds to the default level • `0` is no compression • `1` is the best speed • `9` is the best compression	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.compression-level=[-1-9]`
Compression Strategy	Sets the strategy to be used during HTTP compression.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.compression-strategy=[Default/Filtered/Huffman Only]`

Configuration settings	Description	Asadmin Command
Compressible Mime Types	Comma separated list of MIME types for which HTTP compression is used.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.compressable-mime-type=value`
Compression Minimum Size	The minimum size of a file that determines whether compression is applied.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.compression-min-size-bytes=value`
No-Compression User Agents	Comma separated list of regular expressions matching user agents for which compression should not be used.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.no-compression-user-agents=value`
Restricted User Agents	List of restricted user agents for which HTTP compression is applied.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.restricted-user-agents=value`
Default Response Type	The default response type if none are matched. Specified as a semicolon delimited string consisting of content type, encoding, language, charset.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.default-response-type=value`
Adapter	The class name of the static resources adapter.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.adapter=value`
Comet Support	Whether comet support is enabled or not.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.comet-support-enabled=[true|false]`

Configuration settings	Description	Asadmin Command
DNS Lookup	Enables Domain Name System lookups.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.dns-lookup-enabled=[true|false]`
Trace Enabled	Enables the TRACE operation **TIP** Setting this to false helps prevent cross-site scripting attacks.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.trace-enabled=[true|false]`
Auth Pass Through	Enable this if the network listener receives traffic from an SSL terminating proxy server.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.auth-pass-through-enabled=[`true|false]`
Chunking Enabled	Enables HTTP response chunking.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.chunking-enabled=[true|false]`
XFrame Options	When enabled, the `X-Frame-Options` header will be set to `SAMEORIGIN`. This can still be overridden by a deployed application.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.xframe-options=[true|false]`
Encoded Slash	Allow encoded slashes (e.g. `%2F`) in URIs.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.encoded-slash-enabled=[true|false]`
Websockets Support Enabled	Allows Websockets to be used.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.websockets-support-enabled=[true|false]`

Configuration settings	Description	Asadmin Command
Scheme Mapping	HTTP header name used for identifying the originating protocol of an HTTP request.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.scheme-mapping=value`
Remote User Mapping	HTTP header name used for identifying the originating user of an HTTP request.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.remote-user-mapping=value`
Enable Cookie SameSite	Enabling to set all cookies to use the SameSite value.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.cookie-same-site-enabled=[true|false]`

Configuration settings	Description	Asadmin Command
Cookie SameSite Value	The values to be used: • `Lax`: Cookies are not sent on normal cross-site sub-requests • `Strict`: Cookies will only be sent in a first-party context • `None`: Cookies will be sent in all contexts Users can set up the SameSite Cookie attribute at the **application level** by using the `payara-web.xml` deployment descriptor. Set this attribute by using the `cookie-properties` tag with a child property named `cookieSameSite`: `<payara-web-app> <cookie-properties> <property name="cookieSameSite" value="Strict" /> </cookie-properties> </payara-web-app>`	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.cookie-same-site-value=[`Lax`|`Strict`|`None`]`

Table 8. HTTP/2 Settings

Configuration settings	Description	Asadmin Command
HTTP/2 Enabled	Whether to enable HTTP/2 support. **IMPORTANT** | HTTP/2 support can only be enabled on a secure network listener	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.http2-enabled=[true|false]`
Disable HTTP/2 Cipher Check	Whether TLS connections can be established using insecure cipher suites.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.http2-disable-cipher-check=[true|false]`
HTTP/2 Max Concurrent Streams	The number of maximum concurrent streams allowed per HTTP/2 connection.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.http2-max-concurrent-streams=value`
HTTP/2 Initial Window Size	The initial window size in bytes.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.http2-initial-window-size-in-bytes=value`
HTTP/2 Max Frame Payload Size	The maximum size of the HTTP/2 frame payload to be accepted.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.http2-max-frame-payload-size-in-bytes=value`
HTTP/2 Max Header List Size	The maximum size, in bytes, of the header list.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.http2-max-header-list-size-in-bytes=value`

Configuration settings	Description	Asadmin Command
Streams High Watermark	The maximum number of streams that are allowed. Excess streams will be cleaned out periodically in proportion to the maximum number of streams.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.http2-streams-high-water-mark=value`
Streams Clean Percentage	The number of streams to process when the high watermark is exceeded. **NOTE** | Only closed streams will be removed.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.http2-clean-percentage=value`
Streams Clean Frequency Check	The number of streams that must be closed before checking if the number of streams exceeds the high watermark.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.http2-clean-frequency-check=value`
HTTP/2 Push	Enables server push. This is enabled by default.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.http.http2-push-enabled=[true|false]`

Table 9. HTTP/2 Settings

Configuration settings	Description	Asadmin Command
TLS V1.2 Enabled	Whether to enable TLS V1.2 or not. Will be ignored if TLS is disabled.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.tls12-enabled=[true|false]`

Configuration settings	Description	Asadmin Command
TLS V1.3 Enabled	Whether to enable TLS V1.3 or not. Will be ignored if TLS is disabled. **NOTE** Support for TLS 1.3 is available with JDK 8 versions after JDK 1.8.0u261, unless you're using Zulu JDK 1.8.0u222 - 1.8.0u252, in which case you'll also need to use the Java option -XX:+UseOpenJSSE. This option makes OpenJSSE default TLS provider. OpenJSEE is a JSEE provider created by Azul to support TLS 1.3 on JDK 8. See TLS 1.3 Support in Zulu 8 with OpenJSSE for more information. Also note that this flag can cause problems with the HTTP/2 support. If you're using a lower version than 1.8.0u222, checkbox to enable TLS 1.3 will not be visible on the web administration console.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.tls13-enabled=[true|false]`

Configuration settings	Description	Asadmin Command
Client Authentication	When enabled, clients will be required to authenticate themselves to the server.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.client-auth-enabled=[true|false]`
Certificate Nickname	The alias of the certificate to be supplied on secure requests. The certificate should be present in the server keystore.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.cert-nickname=value`
Key Store	The name of the keystore file used to secure communication handled by this network listener.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.key-store=value`
Trust Store	The name of the trust store file where trusted certificates are stored.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.trust-store=value`
Trust Algorithm	The name of the trust management algorithm (e.g. PKIX) to use for certification path validation.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.trust-algorithm=value`
Max Certificate Length	Maximum number of non self-issued intermediate certificates that can exist in a certification path. Only used if the trust algorithm is set to PKIX.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.trust-max-cert-length=value`
Enabled Ciphers	A list of the server enabled ciphers. This string is stores as a comma separated list of the enabled ciphers with a `+` or a `-` at the start of each, depending on whether the cipher is enabled or disabled. E.g. `+TLS_RSA_WITH_AES_256_CBC_SHA,+SSL_RSA_WITH_3DES_EDE_CBC_SHA`.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.ssl3-tls-ciphers=value`

Configuration settings	Description	Asadmin Command
Handshake Timeout	The timeout, in milliseconds, for a handshake. After this timeout the handshake will be aborted.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.handshake-timeout-millis=value`
TLS Rollback	Whether TLS rollback is enabled or not.	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.tls-rollback-enabled=[true|false]`

<table>
<tr><th>Configuration settings</th><th>Description</th><th>Asadmin Command</th></tr>
<tr><td>HSTS Enabled</td><td>Whether HSTS is enabled. When enabled, the server will respond to requests with the <code>Strict-Transport-Security</code> header. While these requests often return a 301 directing the client to the secure site, this header instructs the requester to make all connections to this site secure for the next year.

The header takes the following form:

<pre>Strict-Transport-Security: max-age=31536000; includeSubDomains; preload</pre>
The last 2 attributes are added based on the other HSTS configuration options.

NOTE
HSTS can be enabled on any HTTP listener. However, it’s meant to be enabled on a secure HTTP listener if an insecure HTTP listener redirects to the secure listener. When browsers receive the <code>Strict-Transport-Security</code> header over HTTPS, they will use HTTPS immediately for all future connections requested over plain HTTP.</td><td><code>set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.hsts-enabled=[true|false]</code>

NOTE
When the configuration value is set to true, the HSTS header is added with a max-age property value of 31536000. The max-age property is not configurable.</td></tr>
</table>

Configuration settings	Description	Asadmin Command
HSTS Subdomains	When enabled in combination with HSTS, `Strict-Transport-Security` headers will include the includeSubDomains attribute. When this is configured, the client will also assume subdomains of the targeted resource require secure connections. The subdomains property is not enabled by default and has to be enabled with the following command:	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.hsts-sub-domains=[true|false]`
HSTS Preload	When enabled in combination with HSTS, `Strict-Transport-Security` headers will include the `preload` attribute. When this is configured, the domain of the target resource will be added to the browser `preload` list, meaning that initial requests to this resource in future won't be insecure. The `preload` property is not enabled by default and has to be enabled with the following command:	`set configs.config.server-config.network-config.protocols.protocol.${protocol-name}.ssl.hsts-preload=[true|false]`

Create a Protocol

Use the `create-protocol` subcommand in remote mode to create a protocol.

1. Ensure that the server is running.

   Remote subcommands require a running server.

2. Create a protocol by using the "`create-protocol`"

   Information about options and properties for the subcommand are included in this help page.

Example - Creating an HTTP Protocol

This example creates a protocol named `http-1` with security enabled.

```
asadmin> create-protocol --securityenabled=true http-1
Command create-protocol executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-protocol` at the command line.

List Protocols

Use the `list-protocols` subcommand in remote mode to list the existing HTTP protocols.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the existing protocols by using the "`list-protocols`" subcommand.

Example - Listing the Protocols

This example lists the existing protocols.

```
asadmin list-protocols
admin-listener
http-1
http-listener-1
http-listener-2
Command list-protocols executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-protocols` at the command line.

Delete a Protocol

Use the `delete-protocol` subcommand in remote mode to remove a protocol.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Delete a protocol by using the "`delete-protocol`" subcommand

Example - Deleting a Protocol

This example deletes the protocol named `http-1`.

```
asadmin delete-protocol http-1
Command delete-protocol executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-protocol` at the command line.

Administering HTTP Configurations

Each HTTP listener has an HTTP configuration, which is created either by using the `create-http` subcommand or by using the built-in configurations that are applied when you follow the instructions in to-create-an-http-network-listener.

Create an HTTP Configuration

Use the `create-http` subcommand in remote mode to create a set of HTTP parameters for a protocol. This set of parameters configures one or more network listeners,

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create an HTTP configuration by using the "`create-http`" subcommand.

   Information about options and properties for the subcommand are included in this help page.

Example - Creating an HTTP Configuration

This example creates an HTTP parameter set for the protocol named `http-1`.

```
asadmin create-http --timeout-seconds 60 --default-virtual-server server http-1
Command create-http executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-http` at the command line.

Delete an HTTP Configuration

Use the `delete-http` subcommand in remote mode to remove HTTP parameters from a protocol.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Delete the HTTP parameters from a protocol by using the "`delete-http`" subcommand.

Example - Deleting an HTTP Configuration

This example deletes the HTTP parameter set from a protocol named `http-1`.

```
asadmin delete-http http-1
Command delete-http executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-http` at the command line.

Administering HTTP Transports

Each HTTP listener has an HTTP transport, which is created either by using the `create-transport` subcommand or by using the built-in transports that are applied when you follow the instructions in to-create-an-http-network-listener.

Create a Transport

Use the `create-transport` subcommand in remote mode to create a transport for a network listener,

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create a transport by using the "`create-transport`" subcommand.

   Information about options and properties for the subcommand are included in this help page.

Example - Creating a Transport

This example creates a transport named `http1-trans` that uses a non-default number of acceptor threads.

```
asadmin create-transport --acceptorthreads 100 http1-trans
Command create-transport executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-transport` at the command line.

List Transports

Use the `list-transports` subcommand in remote mode to list the existing HTTP transports.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the existing transports by using the "`list-transports`" subcommand.

Example - Listing HTTP Transports

This example lists the existing transports.

```
asadmin list-transports
Enter the value for the target operand> server
http1-trans
tcp
Command list-transports executed successfully.
```

The values for target operand are as follows:

target: Restricts the listing to transports for a specified target. Valid values are as follows:
`server`;; Lists the transports for the default server instance. This is the default value. configuration-name;; Lists the transports for the specified configuration. cluster-name;; Lists the transports for all server instances in the specified cluster. instance-name;; Lists the transports for the specified server instance. You can also view the full syntax and options of the subcommand by typing `asadmin help list-transports` at the command line.

Delete a Transport

Use the `delete-transport` subcommand in remote mode to remove a transport.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Delete a transport by using the "`delete-transport`" subcommand.

Example - Deleting a Transport

This example deletes the transport named `http1-trans`.

```
asadmin delete-transport http1-trans
Command delete-transport executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-transport` at the command line.

Administering Virtual Servers

A virtual server is a virtual web server that serves content targeted for a specific URL. Multiple virtual servers can serve content using the same or different host names, port numbers, or IP addresses. The HTTP service directs incoming web requests to different virtual servers based on the URL.

Just as multiple network cards in a single PC make the network see them as separate hosts, virtual servers allow one physical server instance to appear as many, enabling multiple domain names to be associated with the same server.

When you first install Payara Server, a default virtual server is created. You can assign a default virtual server to each new HTTP listener you create.

Web applications and Jakarta EE applications containing web components (web modules) can be assigned to virtual servers during deployment. A web module can be assigned to more than one virtual server, and a virtual server can have more than one web module assigned to it. If you deploy a web application and don’t specify any assigned virtual servers, the web application is assigned to all currently defined virtual servers.

If you then create additional virtual servers and want to assign existing web applications to them, you must redeploy the web applications. For more information about deployment, see the "Payara Server Application Deployment section".

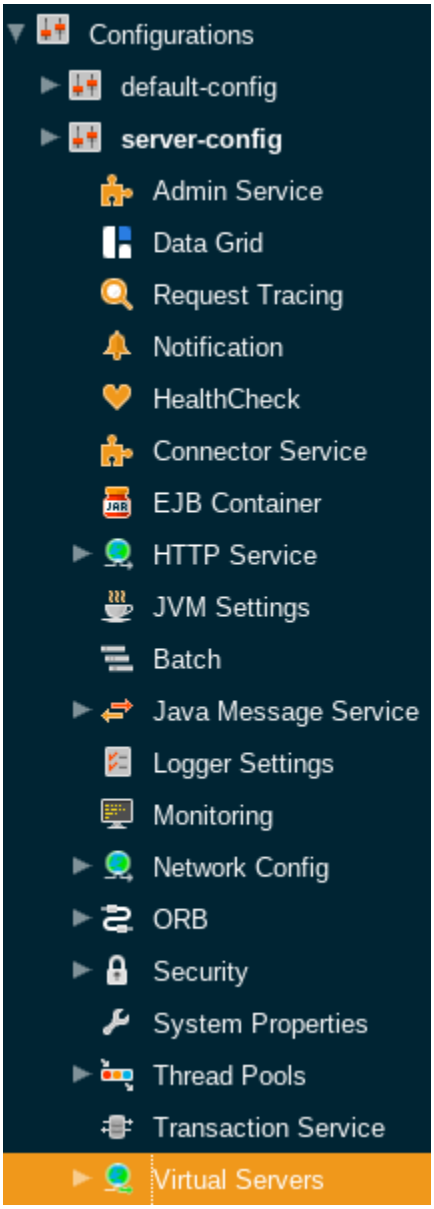
You can define virtual server properties using the `asadmin set` command. For example:

```
asadmin set server-config.http-service.virtual-server.MyVS.property.sso-enabled
="true"
```

Some virtual server properties can be set for a specific web application. For details, see ""payara-web-app"" in the Payara Server Application Deployment section.

Virtual Server Configuration

Existing virtual servers can be configured from the admin console or with `set` asadmin commands. The virtual server configurations can be found in the admin console under **Configurations** ▯ **your-config** ▯ **Virtual Servers**.



The following fields can be configured:

Table 10. Virtual Server Configuration

Configuration settings	Description	Asadmin Command
ID	The unique ID for the virtual server. The only field which cannot be altered from the admin console. The ID cannot begin with a number.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.id=value`
Hosts	A comma separated list of values allowed in the host request header to select this virtual server. Each virtual server with the same connection group must have a unique host for that group.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.hosts=value,value2`
State	The enabled state of the virtual server. There are 3 options for this property: • **On** This means that the virtual server is active and will be ready to serve requests. • **Off** This means that the virtual server is inactive, and will return a 404 error for all requests made through it. This will look from the outside as if no applications are deployed to this server. • **Disabled** This means that the virtual server is inactive, and will return a 403 error for all requests made through it. This will look from the outside as if the server is intentionally down for maintenance.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.state=on/off/disabled`

Configuration settings	Description	Asadmin Command
SSO	Allows multiple web applications deployed to the same virtual server to share the user authentication state. This would mean that signing in to one application would implicitly sign you into all others. There are 3 options for this property: • **True** (enabled) – This enables SSO for applications on the same virtual server. • **False** (disabled) – This disables SSO for applications on the same virtual server. • **Inherit** (controlled by HTTP service) – This will inherit from the value specified in the **HTTP Service**.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.sso-enabled=true/false/inherit`
SSO Cookie Http Only	Determines whether to add the `HttpOnly` flag to the `JSESSIONIDSSO` cookie. In practice, this disallows JavaScript access to the cookie in the browser. This prevents malicious XSS attacks hijacking a Java session. It is enabled by default.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.sso-cookie-http-only=true/false`

Configuration settings	Description	Asadmin Command
Session Cookie Http Only	Determines whether to add the `HttpOnly` flag to the `JSESSIONID` cookie. In practice, this disallows JavaScript access to the cookie in the browser. This prevents malicious XSS attacks hijacking a Java session. It is enabled by default.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.session-cookie-http-only=true/false`
Session Cookie Secure	Determines whether to add the `Secure` flag to the `JSESSIONID` cookie. In practice, this specifies that the cookie may only be transmitted using HTTPS connections.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.session-cookie-secure=true/false/dynamic`
Network Listeners	Comma separated list of network listeners which applications deployed to this virtual server are available on.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.network-listeners=listener-1,listener-2`
Default Web Module	This defines which application deployed to this virtual server will be requested if no modules are resolved in the request. For example if `app1` and `app2` have context roots of `/app1` and `/app2` respectively, `app1` will be returned when the user requests `/`.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.default-web-module=app1`
Log File	This defines a file where log messages coming from this virtual server are redirected to. This allows you to separate application logs from server logs.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.log-file=/dir/to/logs/file.log`

Configuration settings	Description	Asadmin Command
Docroot	This defines the directory used as the document root for this virtual server. A document root is a directory where static files can be placed and served from. For example, if you have a web application deployed to `http://localhost:8080/app1` and the corresponding virtual server has a docroot value of `/opt/server/docs`, then requesting `http://localhost:8080/app1/img.jpg` will fetch the file at `/opt/server/docs/img.jpg`.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.docroot=/dir/to/docs/`
Access Logging	Enables access logging for this virtual server. Access logging logs all requests made to that virtual server. There are 3 options for this property: • **True** (enabled) – This enables access logging for applications on the same virtual server. • **False** (disabled) – This disables access logging for applications on the same virtual server. • **Inherit** (controlled by HTTP service) – This will inherit from the value specified in the **HTTP Service**.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.access-logging=true/false/inherit`

Configuration settings	Description	Asadmin Command
Access Log Directory	Defines the directory used for access logs.	`set configs.config.server-config.http-service.virtual-server.${virtual-server-name}.access-log=/dir/to/logs`

Additional Properties

Defines additional properties for the configuration of a virtual server.

Key	Description	Default Value
sso-max-inactive-seconds	The number of seconds of no activity after which a user's SSO record becomes eligible for purging.	`300`
sso-reap-interval-seconds	The number of seconds between purges of SSO records.	`60`
setCacheControl	Comma separated list of Cache-Control response directives (See section 14.9 of: https://www.ietf.org/rfc/rfc2616.txt).	none

allowLinking	If set to `true`, resources that are symbolic links will be served for all applications deployed to this virtual server.
	
	Can be overridden for an individual application in the `glassfish-web.xml` / `payara-web.xml` deployment descriptor with:
	
	`<payara-web-app>`
	`  <property name="allowLinking" value="false" />`
	`</payara-web-app>`
	
	IMPORTANT glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.
	
	NOTE Setting this property to `true` on Windows systems exposes JSP source code.
		false

Key	Description	Default Value
accessLogWriteInterval	The number of seconds between each time the log is written to disk. The access log is written when the buffer is full or when the interval expires. If the value is 0, the buffer is always written even if it's not full.	300
accessLogBufferSize	The size, in bytes, of the buffer where access log calls are stored.	32768
allowRemoteAddress	Comma separated list of regular expression patterns to which the remote client's IP address is compared. The IP must match one of the patterns to be accepted.	none
denyRemoteAddress	Comma separated list of regular expression patterns to which the remote client's IP address is compared. The IP must not match any of the patterns to be accepted.	none
allowRemoteHost	Comma separated list of regular expression patterns to which the remote client's host name is compared. The host name must match one of the patterns to be accepted.	none
denyRemoteHost	Comma separated list of regular expression patterns to which the remote client's host name is compared. The host name must not match any of the patterns to be accepted.	none
authRealm	Specifies the name attribute of a security realm, which overrides the server's default realm for standalone web applications deployed to this virtual server. Can be overridden for an individual application with a realm defined in the web.xml.	none
securePagesWithPragma	Set this property to `false` to ensure that for all web applications on this server file downloads using SSL work properly in Internet Explorer.	true
contextXmlDefault	Sets the directory of the `context.xml` used for this virtual server, if one is used.	none
alternatedocroot_n	Specifies an alternate document root, where n is a positive integer that allows specification of more than one.	none

Key	Description	Default Value
send-error_n	Specifies custom error page mappings for the virtual server, which are inherited by all web applications deployed on the virtual server, unless overridden in the web.xml. The value is a string, composed of three values: • code – The HTTP response status code for the custom error page. • path – The path of the file to be returned, relative to the `/config` directory. • reason (optional) – The text of the reason string. E.g. `send-error_1="code=404 path=/dir/of/file/error.html reason=UNAUTHORIZED"`.	none
redirect_n	Specifies a redirect from an old url to a new one. The value is a string, composed of two values: • `from` – The prefix of the requested URI to match. • `url-prefix` – The prefix of the new URI to return to the client. The **from** prefix is replaced by this value. E.g. `redirect_1="from=/fake url-prefix=http://another"`.	none
listener_n	Specifies the fully qualified class name of a custom Catalina listener, where n is a positive integer that allows the specification of more than one. The listener class must implement the `org.apache.catalina.ContainerListener` or `org.apache.catalina.LifecycleListener` interface.	none
errorReportValve	Specifies the fully qualified class name of a custom valve that produces default error pages for applications on this virtual server. Specify an empty string to disable the default error page mechanism for this virtual server.	`org.apache.catalina.valves.ErrorReportValve`

Key	Description	Default Value
cacheTTL	Specifies the time to live (TTL), expressed in milliseconds, for the virtual server cache used to serve files.	5000

Create a Virtual Server

By default, when Payara Server starts, the following virtual servers are started automatically:

- A virtual server named `server`, which hosts all user-defined web modules. For development, testing, and deployment of web services in a non-production environment, `server` is often the only virtual server required.
- A virtual server named `__asadmin`, which hosts all administration-related web modules (specifically, the Administration Console). This server is restricted, which means that you cannot deploy web modules to this virtual server.

In a production environment, additional virtual servers provide hosting facilities for users and customers so that each appears to have its own web server, even though there is only one physical server.

Use the `create-virtual-server` subcommand in remote mode to create the named virtual server.

Before You Begin

A virtual server must specify an existing HTTP listener. Because the virtual server cannot specify an HTTP listener that is already being used by another virtual server, create at least one HTTP listener before creating a new virtual server.

1. Ensure that the server is running.

   Remote subcommands require a running server.

2. Create a virtual server by using the "`create-virtual-server`" subcommand.

Information about properties for this subcommand is included in this help page.

3. To apply your changes, restart Payara Server.

   See to-restart-a-domain.

Example - Creating a Virtual Server

This example creates a virtual server named `sampleServer` on `localhost`.

```
asadmin create-virtual-server --hosts pigeon,localhost --property authRealm=ldap
sampleServer
Command create-virtual-server executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-virutal-server` at the command line.

To List Virtual Servers

Use the `list-virtual-servers` subcommand in remote mode to list the existing virtual servers.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List virtual servers by using the "`list-virtual-servers`" subcommand.

Example - Listing Virtual Servers

This example lists the virtual servers for `localhost`.

```
asadmin list-virtual-servers

sampleListener
admin-listener
http-listener-2
http-listener-1
Command list-http-listeners executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-virutal-servers` at the command line.

Update a Virtual Server

1. List virtual servers by using the "`list-virtual-servers`" subcommand.
2. Modify the values for the specified virtual server by using the "`set`" subcommand.

   The virtual server is identified by its dotted name.

Delete a Virtual Server

Use the `delete-virtual-server` subcommand in remote mode to delete an existing virtual server.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List virtual servers by using the "`list-virtual-servers`" subcommand.
3. If necessary, notify users that the virtual server is being deleted.
4. Delete a virtual server by using the "`delete-virtual-server`" subcommand.
5. To apply your changes, restart Payara Server. See to-restart-a-domain.

Example - Deleting a Virtual Server

This example deletes the virtual server named `sampleServer` from `localhost`.

```
asadmin> delete-virtual-server sampleServer
Command delete-virtual-server executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-virutal-server` at the command line.

To Assign a Default Web Module to a Virtual Server

A default web module can be assigned to the default virtual server and to each new virtual server. To access the default web module for a virtual server, point the browser to the URL for the virtual server, but do not supply a context root.

For example:

```
http://myvserver:3184/
```

A virtual server with no default web module assigned serves HTML or JavaServer Pages

(JSP) content from its document root, which is usually `domain-dir/docroot`.

To access this HTML or JSP content, point your browser to the URL for the virtual server, do not supply a context root, but specify the target file.

For example:

```
http://myvserver:3184/hellothere.jsp
```

Assign a Virtual Server to an Application or Module

You can assign a virtual server to a deployed application or web module.

Before You Begin

The application or module must already be deployed.

1. In the Administration Console, open the HTTP Service component under the relevant configuration.
2. Open the Virtual Servers component under the HTTP Service component.
3. Select the virtual server to which you want to assign a default web module.
4. Select the application or web module from the Default Web Module drop-down list.

To Set `JSESSIONIDSSO` Cookie Attributes

Use the `sso-cookie-http-only` and `sso-cookie-secure` virtual server attributes to set the `HttpOnly` and `Secure` attributes of any `JSESSIONIDSSO` cookies associated with web applications deployed to the virtual server.

Use the "`set`" subcommand to set the value of the `sso-cookie-http-only` and `sso-cookie-secure` virtual server attributes.

The values supported for these attributes are as follows:

`sso-cookie-http-only`: A boolean value that specifies whether the `HttpOnly` attribute is included in `JSESSIONIDSSO` cookies.

+ When set to `true`, which is the default, the `HttpOnly` attribute is included. When set to `false`, the `HttpOnly` attribute is not included. **`sso-cookie-secure`**: A string value that specifies whether the `Secure` attribute is included in `JSESSIONIDSSO` cookies. Allowed values are as follows:

+ * `true` — The `Secure` attribute is included. * `false` — The `Secure` attribute is not included. * `dynamic` — The `Secure` attribute setting is inherited from the default ports table for all HTTP listeners first session participating in SSO. This is the default value.

To Set `JSESSIONID` Cookie Attributes

Use the `session-cookie-http-only` and `session-cookie-secure` virtual server attributes to set the `HttpOnly` and `Secure` attributes of any `JSESSIONID` cookies associated with web applications deployed to the virtual server.

Use the "`set`" subcommand to set the value of the `session-cookie-http-only` and `session-cookie-secure` virtual server attributes.

The values supported for these attributes are as follows:

`session-cookie-http-only`: A boolean value that specifies whether the `HttpOnly` attribute is included in `JSESSIONID` cookies.

+ When set to `true`, which is the default, the `HttpOnly` attribute is included. When set to `false`, the `HttpOnly` attribute is not included. **`session-cookie-secure`**: A string value that specifies whether the `Secure` attribute is included in `JSESSIONID` cookies. Allowed values are as follows:

+ * `true` — The `Secure` attribute is included. * `false` — The `Secure` attribute is not included. * `dynamic` — The `Secure` attribute setting depends on whether the HTTP connection is secure. This is the default value.

HTTP Header Fields Validation

By default, The following characters are stipulated to be dangerous to parse in both the name and value of all headers passed down on an HTTP request:

Character	Representation
NUL character	This character is expressed in the following forms on a literal value: `\0` `\x00` review explanation here
LF Character (new line)	This character is expressed in the following forms on a literal value: `\n` `\x0A` review explanation here
CR Character (carriage return)	This character is expressed in the following forms on a literal value: `\r` `\x0D` review explanation here

As stipulated in section 5.5 Field Values of the RFC-9110 specification, these characters are deemed invalid and as such the server, as recipient of the request WILL reject it automatically upon detecting the presence of these characters.

NOTE

On previous RFC specifications that detailed HTTP 1.1 semantics, these characters were considered **acceptable**, however from RFC-9110 onwards, the use of these characters is no longer acceptable for header fields names and values.

In some cases, to support backwards compatibility it might be necessary to allow these characters to be processed by the server runtime. In these instances, the following Grizzly (HTTP) system properties allow header fields character validation to be turned on/off:

Table 11. Grizzly Header Field Validation Properties

Property	Description
`org.glassfish.grizzly.http.STRICT_HEADER_NAME_VALIDATION_RFC_9110`	This property enables the validation of all header field names on incoming HTTP requests to prevent any usage of invalid characters on it. When set to `true`, the presence of these characters will trigger the server to invalidate the request with a `400 - Bad Request` response. The default value is `true`.

Property	Description
`org.glassfish.grizzly.http.STRICT_HEADER_VALUE_VALIDATION_RFC_9110`	This property enables the validation of all header field values on incoming HTTP requests to prevent any usage of invalid characters on it. When set to `true`, the presence of these characters will trigger the server to invalidate the request with a `400 - Bad Request` response. The default value is `true`.

Administering Concurrent Resources

This chapter provides procedures for administering concurrent resources in the Payara Server environment by using the `asadmin` command-line utility.

Instructions for accomplishing these tasks by using the Administration Console are contained in the Administration Console online help.

About Concurrent Resources

Concurrent resources are managed objects that provide concurrency capabilities to Jakarta EE applications. In Payara Server, you configure concurrent resources and make them available for use by application components such as servlets and EJBs. Concurrent resources are accessed through JNDI lookup or resource injection.

Concurrent resources are resources of the following types:

- Context services
- Managed thread factories
- Managed executor services
- Managed scheduled executor services

For detailed information about concurrent resources, see "Concurrency Specs" in the Jakarta EE specifications.

Default Concurrent Resources

When you create a concurrent resource, you specify a unique JNDI name for the resource. Applications use this name to access the resource.

The Jakarta EE standard specifies that certain default resources be made available to applications, and defines specific JNDI names for these default resources. Payara Server makes these names available through the use of logical JNDI names, which map Jakarta EE standard JNDI names to specific Payara Server resources. For concurrent resources, the mappings are as follows:

java:comp/DefaultContextService

This Jakarta EE standard name is mapped to the `concurrent/__defaultContextService` resource.

java:comp/DefaultManagedThreadFactory

This Jakarta EE standard name is mapped to the `concurrent/__defaultManagedThreadFactory` resource.

java:comp/DefaultManagedExecutorService

This Jakarta EE standard name is mapped to the `concurrent/__defaultManagedExecutorService` resource.

java:comp/DefaultManagedScheduledExecutorService

This Jakarta EE standard name is mapped to the `concurrent/__defaultManagedScheduledExecutorService` resource.

Configuring Context Services

Context services are used to create dynamic proxy objects that capture the context of a container and enable applications to run within that context at a later time. The context of the container is propagated to the thread executing the task.

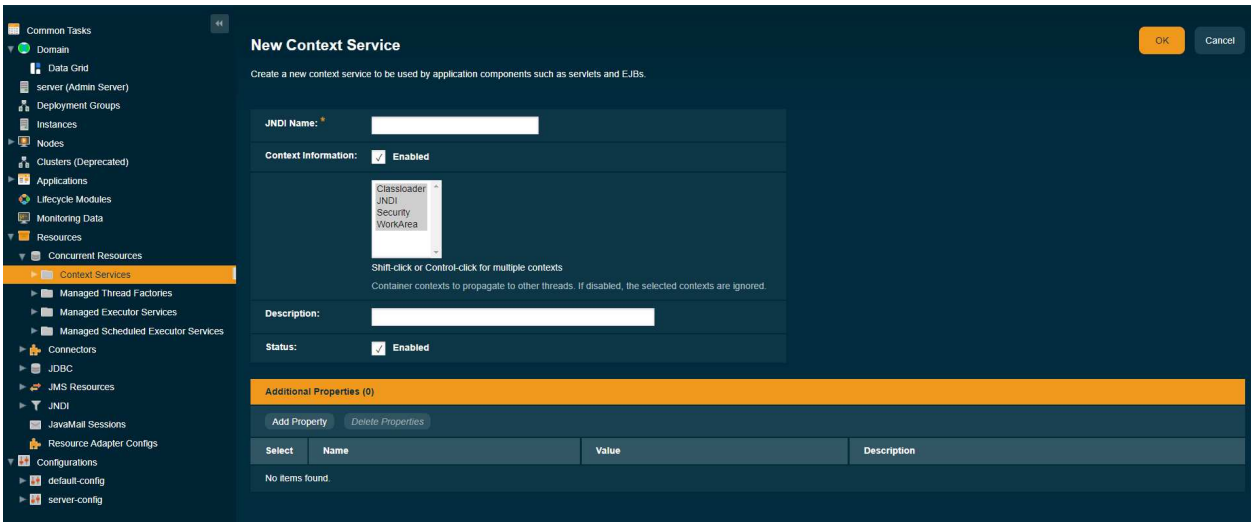
NOTE	Jakarta Concurrency specifies that certain default resources be made available to applications, and defines specific JNDI names for these default resources. Payara Server makes these names available through

> the use of logical JNDI names, which map Jakarta EE JNDI names to specific Payara Server resources. For context services, the mapping is as follows:
> java:comp/DefaultContextService
> This Jakarta Concurrency standard name is mapped to the concurrent/__defaultContextService resource

To Create a Context Service

Context services can be created by using Admin Console or Asadmin commands.

To create a new Context Service in the Admin Console, go to Resources Concurrent Resources Context Services New:



Alternatively, you can use the `create-context-service` asadmin command to create a context service resource.

Because all JNDI names are in the `java:comp/env` subcontext, when specifying the JNDI name of a context service, use only the `concurrent/`name format. For example, `concurrent/Context1`.

IMPORTANT	Creating a context service resource is a dynamic event and typically does not require server restart. Applications can use a resource as soon as it is created. However, if an application tried

> to use a resource before it was created, and that resource is created later, the application or the server must be restarted. Otherwise, the application will not be able to locate the resource.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create a context service by using the "`create-context-service`" subcommand.
3. If necessary, notify users that the new resource has been created.

Example 14-1 to Create a Context Service

This example creates a context service resource named `concurrent/Context1`.

```
asadmin> create-context-service concurrent/Context1
Context service concurrent/Context1 created successfully.
Command create-context-service executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-context-service` at the command line.

To List Context Services

Use the `list-context-services` subcommand in remote mode to list the existing context service resources.

1. Ensure that the server is running.
   Remote subcommands require a running server.
2. List context service resources by using the "`list-context-services`" subcommand.

Example 14-2 to List a Context Service

This example lists context service resources on the default server instance, `server`.

```
asadmin> list-context-services
concurrent/__defaultContextService
concurrent/Context1
concurrent/Context2
Command list-context-services executed successfully.
```

To Update a Context Service

You can change all of the settings for an existing context service resource except its JNDI name. Use the `get` and `set` subcommands to view and change the values of the context service attributes.

IMPORTANT	When a resource is updated, the existing resource is shut down and recreated. If an application used the resource prior to the update, the application or the server must be restarted.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the context service resources by using the "`list-context-services`" subcommand.
3. View the attributes of a specific context service by using the `get` subcommand.
   For example:

```
asdmin> get resources.context-service.concurrent/Context1.*
```

4. Set an attribute of the context service by using the `set` subcommand.
   For example:

```
asdmin> set resources.context-service.concurrent/Context1.deployment-order=120
```

To Delete a Context Service

Use the `delete-context-service` subcommand in remote mode to delete an existing context service. Deleting a context service is a dynamic event and does not require server restart.

Before deleting a context service resource, all associations to the resource must be removed.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the context service resources by using the "`list-context-services`" subcommand.

3. If necessary, notify users that the context service is being deleted.
4. Delete the context service by using the "`delete-context-service`" subcommand.

Example 14-3 to Delete a Context Service

This example deletes the context service resource named `concurrent/Context1`.

```
asadmin> delete-context-service concurrent/Context1
Context service concurrent/Context1 deleted successfully.
Command delete-context-service executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-context-service` at the command line.

Configuring Managed Thread Factories

Managed thread factories are used by applications to create managed threads on demand. The threads are started and managed by the container. The context of the container is propagated to the thread executing the task.

NOTE

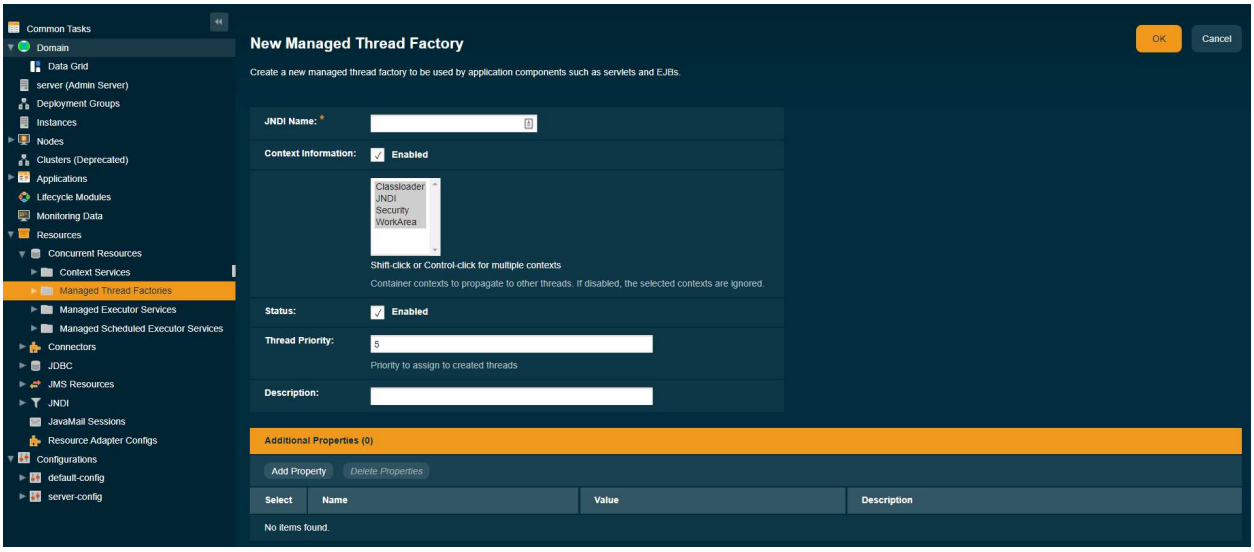
Jakarta Concurrency specifies that certain default resources be made available to applications, and defines specific JNDI names for these default resources. Payara Server makes these names available through the use of logical JNDI names, which map Jakarta EE JNDI names to specific Payara Server resources. For managed thread factories, the mapping is as follows:
java:comp/DefaultManagedThreadFactory
This Jakarta Concurrency standard name is mapped to the concurrent/__defaultManagedThreadFactory resource.

To Create a Managed Thread Factory

Managed Thread Factories can be created by using Admin Console or Asadmin commands.

To create a new Managed Thread Factory in the Admin Console, go to Resources

Concurrent Resources → Managed Thread Factories → New:



Because all JNDI names are in the `java:comp/env` subcontext, when specifying the JNDI name of a managed thread factory, use only the `concurrent/` name format. For example, `concurrent/Factory1`.

For more information about the default managed thread factory resource included with Payara Server, see default-concurrent-resources.

NOTE

Creating a managed thread factory resource is a dynamic event and typically does not require server restart. Applications can use a resource as soon as it is created. However, if an application tried to use a resource before it was created, and that resource is created later, the application or the server must be restarted. Otherwise, the application will not be able to locate the resource.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create a managed thread factory by using the "`create-managed-thread-factory`" subcommand.
3. If necessary, notify users that the new resource has been created.

Example 14-4 to Create a Managed Thread Factory

This example creates a managed thread factory resource named `concurrent/Factory1`.

```
asadmin> create-managed-thread-factory concurrent/Factory1
Managed thread factory concurrent/Factory1 created successfully.
Command create-managed-thread-factory executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-managed-thread-factory` at the command line.

To List Managed Thread Factories

Use the `list-managed-thread-factories` subcommand in remote mode to list the existing managed thread factory resources.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List managed thread factory resources by using the "`list-managed-thread-factories`" subcommand.

Example 14-5 to List a Managed Thread Factory

This example lists managed thread factory resources on the default server instance, `server`.

```
asadmin> list-managed-thread-factories
concurrent/__defaultManagedThreadFactory
concurrent/Factory1
concurrent/Factory2
Command list-managed-thread-factories executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-managed-thread-factories` at the command line.

To Update a Managed Thread Factory

You can change all of the settings for an existing managed thread factory resource except its JNDI name. Use the `get` and `set` subcommands to view and change the values of the managed thread factory attributes.

IMPORTANT	When a resource is updated, the existing resource is shut down and recreated. If applications used the resource prior to the update, the application or the server must be restarted.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the managed thread factory resources by using the "`list-managed-thread-factories`" subcommand.
3. View the attributes of a managed thread factory by using the `get` subcommand. For example:

```
asdmin> get resources.managed-thread-factory.{resource-JNDI-name}.*
```

4. Set an attribute of the managed thread factory by using the `set` subcommand. For example:

```
asdmin> set resources.managed-thread-factory.{resource-JNDI-name}.deployment-
order=120
```

To Delete a Managed Thread Factory

The `delete-managed-thread-factory` asadmin command can be used to delete an existing managed thread factory. Deleting a managed thread factory is a dynamic event and does not require server restart. Before deleting a managed thread factory resource, all associations to the resource must be removed.

Before deleting a managed thread factory resource, all associations to the resource must be removed.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the managed thread factory resources by using the "`list-managed-thread-factories`" subcommand.
3. If necessary, notify users that the managed thread factory is being deleted.
4. Delete the managed thread factory by using the "`delete-managed-thread-`

`factory`" subcommand.

Example 14-6 to Delete a Managed Thread Factory

This example deletes the managed thread factory resource named `concurrent/Factory1`.

```
asadmin> delete-managed-thread-factory concurrent/Factory1
Managed thread factory concurrent/Factory1 deleted successfully.
Command delete-managed-thread-factory executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-managed-thread-factory` at the command line.

Configuring Managed Executor Services

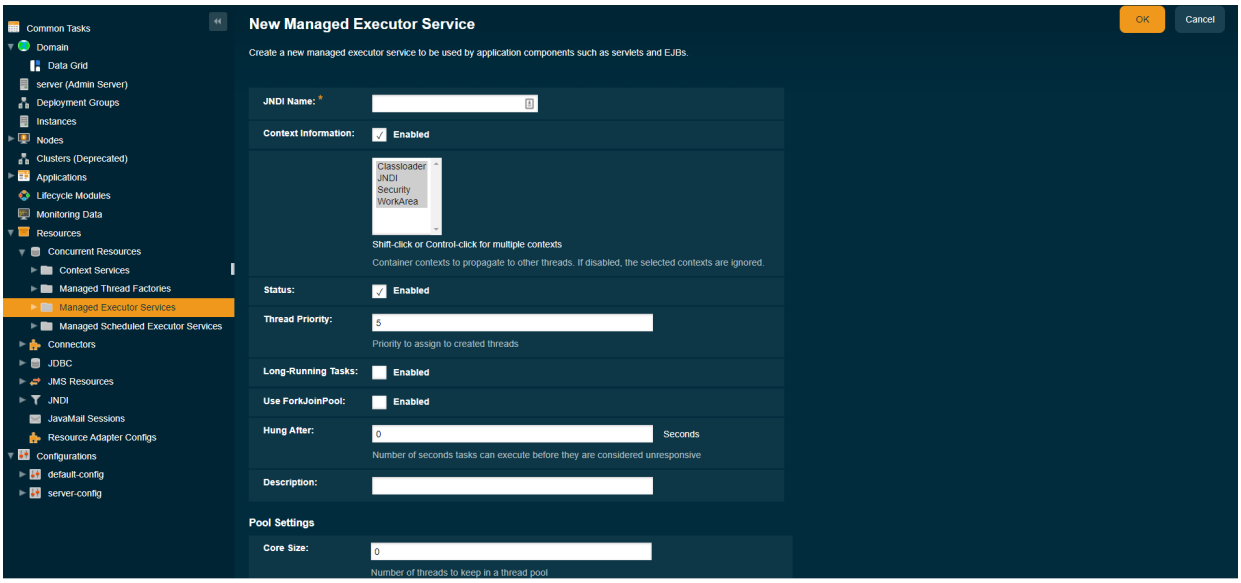
Managed executor services are used by applications to execute submitted tasks asynchronously. Tasks are executed on threads that are started and managed by the container. The context of the container is propagated to the thread executing the task.

IMPORTANT	Jakarta Concurrency specifies that certain default resources be made available to applications, and defines specific JNDI names for these default resources. Payara Server makes these names available through the use of logical JNDI names, which map Jakarta EE JNDI names to specific Payara Server resources. For managed executor services, the mapping is as follows: java:comp/DefaultManagedExecutorService This Jakarta Concurrency standard name is mapped to the concurrent/__defaultManagedExecutorService resource.

To Create a Managed Executor Service

Managed Executor Services can be created by using Admin Console or Asadmin commands.

To create a new Managed Executor Service in the Admin Console, go to Resources Concurrent Resources Managed Executor Services New:



Use the `create-managed-executor-service` subcommand in remote mode to create a managed executor service resource.

Because all JNDI names are in the `java:comp/env` subcontext, when specifying the JNDI name of a managed executor service, use only the `concurrent/`name` format. For example, `concurrent/Executor1`.

For more information about the default managed executor service resource included with Payara Server, see default-concurrent-resources.

NOTE

Creating a managed executor service resource is a dynamic event and typically does not require server restart. Applications can use a resource as soon as it is created. However, if an application tried to use a resource before it was created, and that resource is created later, the application or the server must be restarted. Otherwise, the application will not be able to locate the resource.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create a managed executor service by using the "`create-managed-executor-service`" subcommand.
3. If necessary, notify users that the new resource has been created.

Example 14-7 to Create a Managed Executor Service

This example creates a managed executor service resource named `concurrent/Executor1`.

```
asadmin> create-managed-executor-service concurrent/Executor1
Managed executor service concurrent/Executor1 created successfully.
Command create-managed-executor-service executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-managed-executor-service` at the command line.

To List Managed Executor Services

The `list-managed-executor-services` asadmin command can be used to list the existing managed executor service resources.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List managed executor service resources by using the "`list-managed-executor-services`" subcommand.

Example 14-8 to List a Managed Executor Service

This example lists managed executor service resources on the default server instance, `server`.

```
asadmin> list-managed-executor-services
concurrent/__defaultManagedExecutorService
concurrent/Executor1
concurrent/Executor2
Command list-managed-executor-services executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-managed-executor-services` at the command line.

To Update a Managed Executor Service

You can change all of the settings for an existing managed executor service resource except its JNDI name. Use the `get` and `set` subcommands to view and change the values of the managed executor service attributes respectively.

IMPORTANT	When a resource is updated, the existing resource is shut down and recreated. If applications used the resource prior to the update, the application or the server must be restarted.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the managed executor service resources by using the "`list-managed-executor-services`" subcommand.
3. View the attributes of a managed executor service by using the `get` subcommand. For example:

```
asdmin> get resources.managed-executor-service.{resource-JNDI-name}.*
```

4. Set an attribute of the managed executor service by using the `set` subcommand. For example:

```
asdmin>  set resources.managed-executor-service.{resource-JNDI-name}.deployment-
order=120
```

To Delete a Managed Executor Service

Use the `delete-managed-executor-service` subcommand in remote mode to delete an existing managed executor service. Deleting a managed executor service is a dynamic event and does not require server restart.

Before deleting a managed executor service resource, all associations to the resource must be removed.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the managed executor service resources by using the "`list-managed-executor-services`" subcommand.
3. If necessary, notify users that the managed executor service is being deleted.
4. Delete the managed executor service by using the "`delete-managed-executor-service`" subcommand.

Example 14-9 to Delete a Managed Executor Service

This example deletes the managed executor service resource named `concurrent/Executor1`.

```
asadmin> delete-managed-executor-service concurrent/Executor1
Managed executor service concurrent/Executor1 deleted successfully.
Command delete-managed-executor-service executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-managed-executor-service` at the command line.

Configuring Managed Scheduled Executor Services

Managed scheduled executor services are used by applications to execute submitted tasks asynchronously at specific times. Tasks are executed on threads that are started and managed by the container. The context of the container is propagated to the thread executing the task.

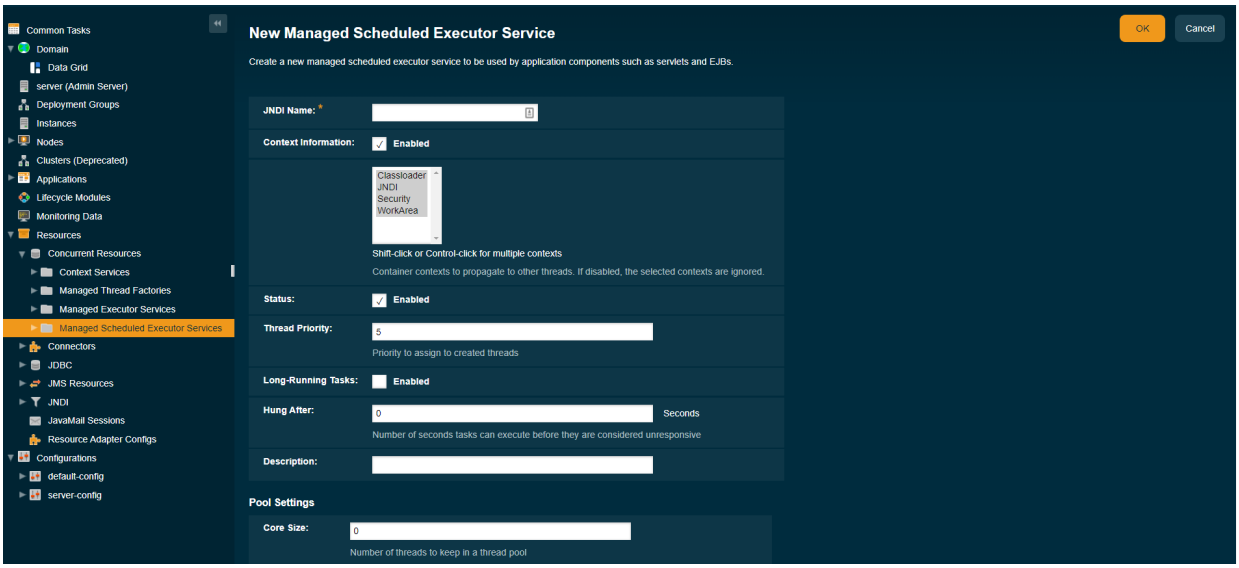
NOTE

Jakarta Concurrency specifies that certain default resources be made available to applications, and defines specific JNDI names for these default resources. Payara Server makes these names available through the use of logical JNDI names, which map Jakarta EE JNDI names to specific Payara Server resources. For managed scheduled executor services, the mapping is as follows:
java:comp/DefaultManagedScheduledExecutorService
This Jakarta Concurrency standard name is mapped to the concurrent/__defaultManagedScheduledExecutorService resource.

To Create a Managed Scheduled Executor Service

Managed Scheduled Executor Services can be created by using Admin Console or Asadmin commands.

To create a new Managed Scheduled Executor Service in the Admin Console, go to Resources Concurrent Resources Managed Scheduled Executor Services New:



Alternatively, the `create-managed-scheduled-executor-service` asadmin command can be used to create a managed scheduled executor service resource.

Because all JNDI names are in the `java:comp/env` subcontext, when specifying the JNDI name of a managed scheduled executor service, use only the `concurrent/`name format. For example, `concurrent/ScheduledExecutor1`.

For more information about the default managed scheduled executor service resource included with Payara Server, see default-concurrent-resources.

NOTE

Creating a managed scheduled executor service resource is a dynamic event and typically does not require server restart. Applications can use a resource as soon as it is created. However, if an application tried to use a resource before it was created, and that resource is created later, the application or the server must be restarted. Otherwise, the application will not be able to locate the resource.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create a managed scheduled executor service by using the "`create-managed-scheduled-executor-service`" subcommand.

3. If necessary, notify users that the new resource has been created.

Example 14-10 to Create a Managed Scheduled Executor Service

This example creates a managed scheduled executor service resource named `concurrent/ScheduledExecutor1`.

```
asadmin> create-managed-scheduled-executor-service concurrent/ScheduledExecutor1
Managed scheduled executor service concurrent/ScheduledExecutor1 created
successfully.
Command create-managed-scheduled-executor-service executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help create-managed-scheduled-executor-service` at the command line.

To List Managed Scheduled Executor Services

Use the `list-managed-scheduled-executor-services` subcommand in remote mode to list the existing managed scheduled executor service resources.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List managed scheduled executor service resources by using the "`list-managed-scheduled-executor-services`" subcommand.

Example 14-11 to List a Managed Scheduled Executor Service

This example lists managed scheduled executor service resources on the default server instance, `server`.

```
asadmin> list-managed-scheduled-executor-services
concurrent/__defaultManagedScheduledExecutorService
concurrent/ScheduledExecutor1
concurrent/ScheduledExecutor2
Command list-managed-scheduled-executor-services executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-managed-scheduled-executor-services` at the command line.

To Update a Managed Scheduled Executor Service

You can change all of the settings for an existing managed scheduled executor service

resource except its JNDI name. Use the `get` and `set` subcommands to view and change the values of the managed scheduled executor service attributes.

IMPORTANT	When a resource is updated, the existing resource is shut down and recreated. If applications used the resource prior to the update, the application or the server must be restarted.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the managed scheduled executor service resources by using the "`list-managed-scheduled-executor-services`" subcommand.
3. View the attributes of a managed scheduled executor service by using the `get` subcommand.
   For example:

```
asdmin>get resources.managed-scheduled-executor-service.{resource-JNDI-name}.*
```

4. Set an attribute of the managed scheduled executor service by using the `set` subcommand.
   For example:

```
asdmin> set resources.managed-scheduled-executor-service.{resource-JNDI-name
}.deployment-order=120
```

To Delete a Managed Scheduled Executor Service

Use the `delete-managed-scheduled-executor-service` subcommand in remote mode to delete an existing managed scheduled executor service. Deleting a managed scheduled executor service is a dynamic event and does not require server restart.

Before deleting a managed scheduled executor service resource, all associations to the resource must be removed.

1. Ensure that the server is running. Remote subcommands require a running server.

2. List the managed scheduled executor service resources by using the "`list-managed-scheduled-executor-service`" subcommand.
3. If necessary, notify users that the managed scheduled executor service is being deleted.
4. Delete the managed scheduled executor service by using the "`delete-managed-scheduled-executor-service`" subcommand.

Example 14-12 to Delete a Managed Scheduled Executor Service

This example deletes the managed scheduled executor service resource named `concurrent/ScheduledExecutor1`.

```
asadmin> delete-managed-scheduled-executor-service concurrent/ScheduledExecutor1
Managed scheduled executor service concurrent/ScheduledExecutor1 deleted
successfully.
Command delete-managed-scheduled-executor-service executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-managed-scheduled-executor-service` at the command line.

Administering the Object Request Broker (ORB)

Payara Server supports a standard set of protocols and formats that ensure interoperability. Among these protocols are those defined by CORBA. The Object Request Broker (ORB) is the central component of CORBA. The ORB provides the required infrastructure to identify and locate objects, handle connection management, deliver data, and request communication. This chapter describes how to configure the ORB and the IIOP listeners.

Instructions for accomplishing the tasks in this chapter by using the Administration Console are contained in the Administration Console online help.

About the ORB

The Common Object Request Broker Architecture (CORBA) model is based on clients requesting services from distributed objects or servers through a well-defined interface by issuing requests to the objects in the form of remote method requests. A remote

method request carries information about the operation that needs to be performed, including the object name (called an object reference) of the service provider and parameters, if any, for the invoked method. CORBA automatically handles network programming tasks such as object registration, object location, object activation, request de-multiplexing, error-handling, marshalling, and operation dispatching.

Configuring the ORB

A CORBA object never talks directly with another. Instead, the object makes requests through a remote stub to the Internet Inter-Orb Protocol (IIOP) running on the local host. The local ORB then passes the request to an ORB on the other host using IIOP. The remote ORB then locates the appropriate object, processes the request, and returns the results.

IIOP can be used as a Remote Method Invocation (RMI) protocol by applications or objects using RMI-IIOP. Remote clients of enterprise beans (EJB modules) communicate with Payara Server by using RMI-IIOP.

Administering IIOP Listeners

An IIOP listener is a listen socket that accepts incoming connections from the remote clients of enterprise beans and from other CORBA-based clients. Multiple IIOP listeners can be configured for Payara Server. For each listener, specify a port number (optional; default 1072), a network address, and security attributes (optional). If you create multiple listeners, you must assign a different port number for each listener.

To Create an IIOP Listener

Use the `create-iiop-listener` subcommand in remote mode to create an IIOP listener.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create an IIOP listener by using the "`create-iiop-listener`" subcommand. Information about the properties for the subcommand is included in this help page.
3. To apply your changes, restart Payara Server. See to-restart-a-domain.

Example 15-1 Creating an IIOP Listener

```
asadmin> create-iiop-listener --listeneraddress 192.168.1.100
--iiopport 1400 sample_iiop_listener
Command create-iiop-listener executed successfully.
```

This example creates an IIOP listener named `sample_iiop_listener`.

To List IIOP Listeners

Use the `list-iiop-listeners` subcommand in remote mode to list the existing IIOP listeners.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the IIOP listeners by using the "`list-iiop-listeners`" subcommand.

Example 15-2 Listing IIOP Listeners

```
asadmin> list-iiop-listeners
orb-listener-1
SSL
SSL_MUTUALAUTH
sample_iiop_listener
Command list-iiop-listeners executed successfully.
```

This example lists all the IIOP listeners for the server instance.

To Update an IIOP Listener

1. List the IIOP listeners by using the "`list-iiop-listeners`" subcommand.
2. Modify the values for the specified IIOP listener by using the "`set`" subcommand.

   The listener is identified by its dotted name.

Example 15-3 Updating an IIOP Listener

```
asadmin> set "server.iiop-service.iiop-listener.SSL.enabled"
server.iiop-service.iiop-listener.SSL.enabled=false
Command set executed successfully.
```

This example changes SSL from enabled to disabled.

To Delete an IIOP Listener

Use the `delete-iiop-listener` subcommand in remote mode to delete an IIOP

listener.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the IIOP listeners by using the "`list-iiop-listeners`" subcommand.
3. Delete an IIOP listener by using the "`delete-iiop-listener`" subcommand.
4. To apply your changes, restart Payara Server. See to-restart-a-domain.

Example 15-4 Deleting an IIOP Listener

```
asadmin> delete-iiop-listener sample_iiop_listener
 Command delete-iiop-listener executed successfully.
```

This example deletes the IIOP listener named `sample_iiop_listener`.

Configuring the New ORB Fragment Processing

The aim of this functionality is to enhance the concurrency mechanism through a new implementation that uses the Lock API. The advantages of this approach instead over relying on synchronized code blocks (default) are:

- Fine-grained control: it uses explicit locking and unlocking, delivering higher accuracy and efficiency.
- Use of condition variables: threads await the fulfillment of specific conditions prior to advancing.
- Multiple object locking: a lock can be shared among various class instances, facilitating more flexible management of synchronization between these instances.

NOTE	The Fragment processing happens when a big message is processed by the ORB. If the ORB message exceeds the default size of 1024 bytes, then the internal fragmentation of the message starts.

To configure the new functionality use the following properties:

Table 12. New ORB Fragment Processing Properties

Property	Description
`com.sun.corba.ee.protocol.enableNewFragmentProcess`	This property enables the new fragment processing. By default, this property is set as `false` to support the default implementation based on synchronized blocks of code.
`com.sun.corba.ee.protocol.newFragmentEmptyConditionTimeout`	You can use this property to change the wait timeout for fragments of a request. By default, the timeout is 10000 milliseconds. This property only takes effect if the `com.sun.corba.ee.protocol.enableNewFragmentProcess` is enabled. The property value is expressed in milliseconds.

Administering the Jakarta Mail Service

Payara Server includes the Jakarta Mail API along with Jakarta Mail service providers that allow an application component to send email notifications over the Internet and to read email from IMAP and POP3 mail servers.

Instructions for accomplishing the tasks in this chapter by using the Administration Console are contained in the Administration Console online help.

About Jakarta Mail

The Jakarta Mail API is a set of abstract APIs that model a mail system. The Jakarta Mail API provides a platform-independent and protocol-independent framework to build mail and messaging applications and provide facilities for reading and sending electronic messages. Service providers implement particular protocols.

Using the API you can add email capabilities to your applications. Jakarta Mail provides access from Java applications to Internet Message Access Protocol (IMAP) and Simple Mail Transfer Protocol (SMTP) capable mail servers on your network or the Internet.

NOTE | The API does not provide mail server functionality; you must have access to a mail server to use Jakarta Mail.

The Jakarta Mail API is implemented as an optional package in the Jakarta platform and is also available as part of the Jakarta EE platform.

To learn more about the Jakarta Mail API, consult the Jakarta Mail specification.

Administering Jakarta Mail Resources

When you create a mail session, the server-side components and applications are enabled to access Jakarta Mail services with JNDI, using the session properties you assign for them. When creating a mail session, you can designate the mail hosts, the transport and store protocols, and the default mail user so that components that use Jakarta Mail do not have to set these properties. Applications that are heavy email users benefit because Payara Server creates a single session object and makes the session available to any component that needs it.

Table 13. Jakarta Mail's settings such as the following can be specified:

Jakarta Mail settings	Description
JNDI Name	The unique name for the mail session. Use the naming sub-context prefix mail/ for Jakarta Mail resources. For example: `mail/MySession`
Mail Host	The host name of the default mail server. The connect methods of the store and transport objects use this value if a protocol-specific host property is not supplied. + The name must be resolvable to an actual host name.
Default User	The default username to provide when connecting to a mail server. The connect methods of the store and transport objects use this value if a protocol-specific username property is not supplied.
Default Return Address	The email address of the default user, in the form: `username@host.domain`.
Description	A descriptive statement for the component.
Session	Indicates whether the mail session is enabled or disabled at this time

To Create a Jakarta Mail Resource

Use the `create-javamail-resource` subcommand in remote mode to create a

Jakarta Mail session resource. The JNDI name for a Jakarta Mail session resource customarily includes the mail/ naming subcontext, For example: `mail/MyMailSession.`

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create a Jakarta Mail resource by using the "`create-javamail-resource`" subcommand. Information about the properties for the subcommand is included in this help page.
3. To apply your changes, restart Payara Server. See to-restart-a-domain.

Example - Creating a Jakarta Mail Resource

```
asadmin> create-javamail-resource --mailhost localhost
--mailuser sample --fromaddress sample\@sun\.com mail/MyMailSession
Command create-javamail-resource executed successfully.
```

This example creates a Jakarta Mail resource named `mail/MyMailSession`. The escape character (\) is used in the `--fromaddress` option to distinguish the dot (.) and at sign (@).

To List Javamail Resources

Use the `list-javamail-resources` subcommand in remote mode to list the existing JavaMail session resources.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the Jakarta Mail resources by using the "`list-javamail-resources`" subcommand.

Example - Listing Jakarta Mail Resources

```
asadmin> list-javamail-resources
mail/MyMailSession
Command list-javamail-resources executed successfuly.
```

This example lists the Jakarta Mail resources on `localhost`.

To Update a Jakarta Mail Resource

1. List the Jakarta Mail resources by using the "`list-javamail-resources`" subcommand.
2. Modify the values for the specified Jakarta Mail source by using the "`set`" subcommand. The resource is identified by its dotted name.

Example - Updating a Jakarta Mail Resource

```
asadmin> set server.resources.mail-resource.mail/
MyMailSession.user=joeserver.resources.mail-resource.mail/
MyMailSession.user=joe
Command set executed successfully.
```

This example changes `joeserver` to `joe`.

To Delete a Jakarta Mail Resource

Use the `delete-javamail-resource` subcommands in remote mode to delete a Jakarta Mail session resource.

References to the specified resource must be removed before running the `delete-javamail-resource` subcommands.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the Jakarta Mail resources by using the "`list-javamail-resources`" subcommands.
3. Delete a Jakarta Mail resource by using the "`delete-javamail-resource`" subcommands.
4. To apply your changes, restart Payara Server. to-restart-a-domain.

Example - Deleting a Jakarta Mail Resource

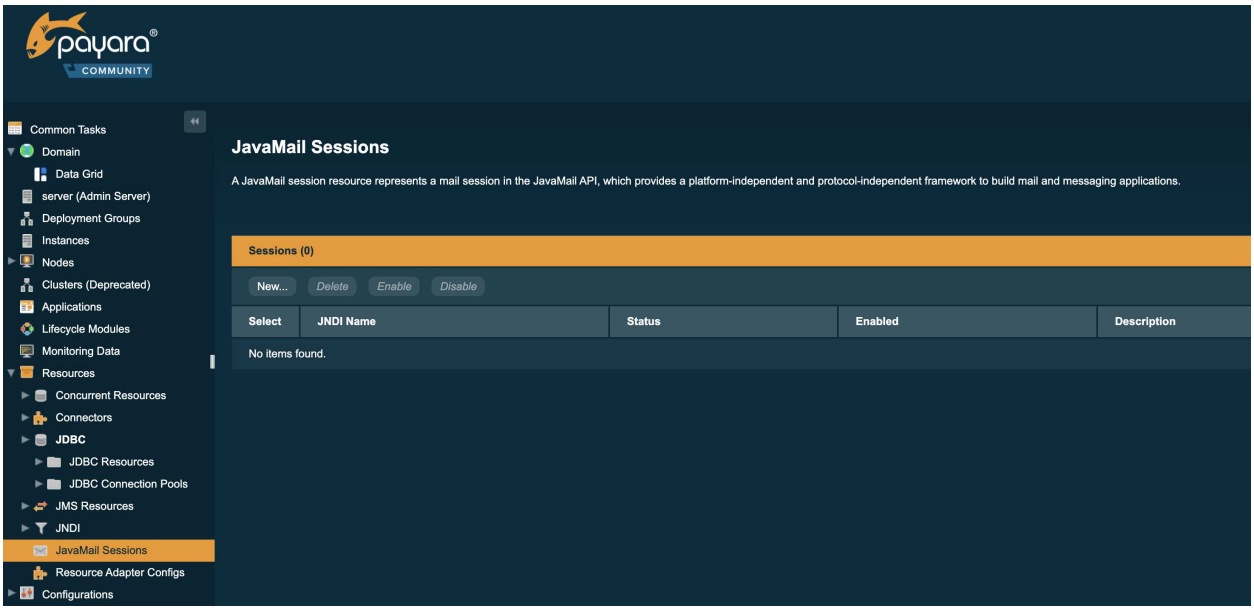
```
asadmin> delete-javamail-resource mail/MyMailSession
Command delete-javamail-resource executed successfully.
```

This example deletes the Jakarta Mail session resource named `mail/MyMailSession`.

Configure a Jakarta Mail Session from the Admin Console

Jakarta Mail sessions are added from the `Resources` tab on the left pane of the Admin Console.

Select `New` on the `Sessions` table to create a new Jakarta Mail session:

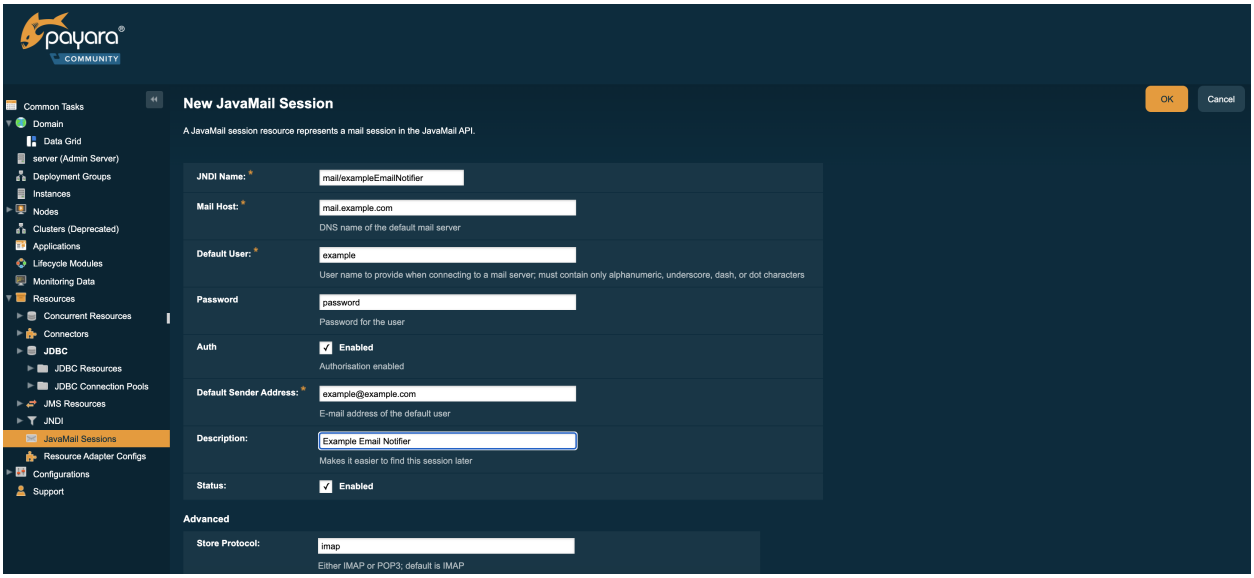


Emails will be sent with a return address of `example@example.com`, using `SMTP` to send mail and `IMAP` to retrieve mail. With the current implementation of the Jakarta Mail API there is no alternative transport protocol provided by default, though as an alternative to the default store protocol of `imap`, `pop3` is available.

Typically, when setting up `SMTP` access, three additional properties must be specified in the Additional Properties table: enabling `startTLS`, enabling authentication, and the password.

These allow for extra security when sending emails and will not always be required - check with your email provider for their specific configuration.

The example below shows the configuration of a mail host:



Administering the Java Message Service (JMS)

The Java Message Service (JMS) API is a messaging standard that allows Jakarta EE applications and components, including message-driven beans (MDBs), to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous.

Payara Server supports JMS messaging by communicating with a JMS provider through a Jakarta EE Connector resource adapter. By default, Payara Server provides JMS messaging through its built-in `jmsra` resource adapter communicating with Open Message Queue, which is included with Payara Server. This combination, known as the JMS Service, is tightly integrated with Payara Server, providing a rich set of `asadmin` subcommands and Administration Console pages to simplify JMS messaging administration tasks.

Instructions for accomplishing the task in this chapter by using the Administration Console are contained in the Administration Console online help.

About the JMS Service

To support JMS messaging, the JMS Service provides the following administrative objects:

- **JMS Service Configuration**: The JMS service configuration is part of the overall

configuration for a Payara standalone instance or cluster. It specifies how the JMS Service is to create and maintain connections with JMS Hosts.

- **JMS Hosts**: JMS hosts are the message servers that host destinations, store messages, and interact with applications to send and receive messages across connections. In Message Queue, JMS hosts are called brokers. + The JMS service supports these types of JMS hosts: + * Embedded type, in which the JMS host runs in the same JVM as the Payara instance; its configuration and lifecycle are managed by the JMS service * Local type, in which the JMS host runs separately on the same host as the Payara instance; its configuration and lifecycle are managed by the JMS service * Remote type, in which the JMS host represents a Message Queue broker or broker cluster that is external to the JMS service; its operation is managed using Message Queue administrative tools + For more information about JMS host types, see about-jms-host-types.
- **JMS Connection Factory Resources**: JMS connection factory resources house the information that applications use to connect to a JMS provider. For each JMS connection factory, the JMS service automatically maintains a Payara connector resource and a Payara connector connection pool in order to support connection pooling and fail-over.
- **JMS Destination Resources**: JMS destination resources house the information that applications use to specify the target destination of messages they produce and the source destination of messages they consume. For each JMS destination resource, the JMS service automatically maintains a Payara administered object.
- **JMS Physical Destinations**: JMS physical destinations provide a means to create and manage JMS destinations administratively instead of having them created dynamically when needed by an application. While dynamic creation of destinations is often sufficient during application development, administratively created physical destinations are more suitable for production environments.

JMS Service High Availability

Just as Payara Server supports clusters of instances to provide high availability,

Message Queue supports clusters of brokers to provide service availability or service and data availability, depending on the type of broker cluster, as described in [Broker Clusters] in Open Message Queue Technical Overview.

The JMS service takes advantage of this Message Queue capability and automatically creates and manages a Message Queue broker cluster when a Payara cluster's configuration specifies Embedded or Local type JMS hosts. Additionally, both Payara clusters and standalone instances can use Message Queue broker clusters as Remote type JMS hosts.

For information about how the JMS service supports Payara clusters and Message Queue broker clusters, see configuring-java-message-service-high-availability in the Payara Server Edition High Availability section.

Updating the JMS Service Configuration

Because the JMS service configuration is part of the overall configuration for a standalone instance or cluster, it is created when the standalone instance or cluster is created. You can then update the JMS service configuration by using the Java Message Service page for the configuration in the Administration Console, or by using a `set` subcommand of the following form:

```
set configs.config.config-name.jms-service.attribute-name=attribute-value
```

The attributes you can set are:

- **`type`**: The JMS host type the service is to use. Available choices are `EMBEDDED`, `LOCAL` and `REMOTE`. See about-jms-host-types for more information.
- **`init-timeout-in-seconds`**: The number of seconds Payara Server waits for the JMS service to start before aborting the startup.
- **`start-args`**: A list of arguments the JMS service passes to Embedded and Local type JMS hosts on startup.
- **`default-jms-host`**: The name of the default JMS host.

- **`reconnect-enabled`**: When set to `true`, the JMS service attempts to reconnect to a JMS host (or one of the JMS hosts in the AddressList) when a connection is lost.
- **`reconnect-attempts`**: The number of attempts to connect (or reconnect) for each JMS host in the AddressList before the JMS service tries the next address in the list. A value of -1 indicates that the number of reconnect attempts is unlimited (the JMS service attempts to connect to the first address until it succeeds).
- **`reconnect-interval-in-seconds`**: The number of seconds between reconnect attempts. This interval applies for attempts on each JMS host in the AddressList and for successive addresses in the list. If it is too short, this time interval does not give a JMS host time to recover. If it is too long, the reconnect might represent an unacceptable delay.
- **`addresslist-behavior`**: The order of connection attempts. Available choices are: + `random`;; Select a JMS host from the AddressList randomly. If there are many clients attempting a connection using the same connection factory, specify `random` to prevent them from all being connected to the same JMS host. `priority`;; Always try to connect to the first JMS host in the AddressList and use another one only if the first one is not available.
- **`addresslist-iterations`**: The number of times the JMS service iterates through the AddressList in an effort to establish (or reestablish) a connection. A value of -1 indicates that the number of attempts is unlimited.
- **`mq-scheme`**:
- **`mq-service`**: The Message Queue address scheme name and connection service name to use for the AddressList if a non-default scheme or service is to be used.

NOTE	After making changes to the JMS service configuration, Payara Server instances that use the configuration must be restarted in order for the changes to be propagated.

Setting Message Queue Broker Properties in the JMS Service Configuration

You can specify any Message Queue broker property in the JMS service configuration

by adding it by name to the Additional Properties table on the Java Message Service page for the configuration in the Administration Console, or by using a `set` subcommand of the following form:

```
set configs.config.config-name.jms-service.property.broker-property-name=value
```

If the broker property name includes dots, preface the dots with two backslashes (`\\`); for example, to set the `imq.system.max_count` property, specify `imq\\.system\\.max_count` in the `set` subcommand.

> **NOTE** You can also set broker properties in the JMS host. If you set the same broker property in both the JMS service configuration and the JMS host, the value specified in the JMS host is used.

Administering JMS Hosts

A JMS host represents a Message Queue broker. JMS contains a JMS hosts list (the `AddressList` property) that contains all the JMS hosts that are used by Payara Server. The JMS hosts list is populated with the hosts and ports of the specified Message Queue brokers and is updated whenever a JMS host configuration changes. When you create JMS resources or deploy message driven beans, the resources or beans inherit the JMS hosts list.

For information about administering JMS hosts that are servicing Payara clusters, see configuring-payara-clusters-to-use-message-queue-broker-clusters in the Payara Server High Availability section.

About JMS Host Types

The JMS service uses Message Queue (MQ) brokers as JMS hosts, integrating them in three ways:

- **Embedded Type**: When the JMS service configuration's `type` attribute is `EMBEDDED`, the MQ broker is co-located in the same JVM as the Payara server instance it services. The JMS service starts it in-process and manages its configuration and

lifecycle. For this type, the JMS service uses lazy initialization to start the broker when the first JMS operation is requested instead of immediately when the Payara instance is started. If necessary, you can force startup of the broker by using the "`jms-ping`" command. Additionally, if the Payara instance is a standalone instance (not a clustered instance), JMS operations use a Message Queue feature called direct mode to bypass the networking stack, leading to performance optimization.

- **Local Type**: When the JMS service configuration's `type` attribute is `LOCAL`, the JMS service starts the MQ broker specified in the configuration as the default JMS host in a separate process on the same host as the Payara server instance. The JMS service manages its configuration and lifecycle. For this type, the JMS service starts the broker immediately when the Payara instance is started. The JMS service provides the Message Queue broker an additional port to start the RMI registry. This port number is equal to the broker's JMS port plus 100. For example, if the JMS port number is 37676, then the additional port's number will be 37776. Additionally, the `start-args` property of the JMS service configuration can be used to specify Message Queue broker startup options.
- **Remote Type**: When the JMS service configuration's `type` attribute is `REMOTE`, the JMS service uses the information defined by the default JMS host to communicate with an MQ broker or broker cluster that has been configured and started using Message Queue tools. Ongoing administration and tuning of the broker or broker cluster are also performed using Message Queue tools.

Configuring Embedded and Local JMS Hosts

Because the JMS service, not Message Queue, manages Embedded and Local JMS hosts automatically, you should avoid using Message Queue utilities to configure them. Instead, specify broker properties in the JMS service configuration or in the JMS host.

Should the need to use Message Queue utilities arise, you must use the `-varhome` option when running certain Message Queue utilities to specify the `IMQ_VARHOME` location of the Embedded or Local JMS host. This location depends on which Payara instance the JMS host is servicing:

- For `server`, the Domain Administration Server (DAS), the `IMQ_VARHOME` location is:

```
domain-root-dir/domain-dir/imq
```

- For any other Payara instance, the `IMQ_VARHOME` location is:

```
as-install/nodes/node-name/instance-name/imq
```

For example, the broker log file for an Embedded or Local JMS host servicing the DAS is available at `domain-root-dir/domain-dir/imq/instances/imqbroker/log/log.txt`, and the broker log file for an Embedded or Local JMS host servicing any other Payara instance is available at `as-install/nodes/node-name/instance-name/imq/instances/mq-instance-name/log/log.txt`.

NOTE

When using Message Queue utilities on the Windows platform, you must explicitly use the Windows executable (`.exe`) versions of the utilities, even when running command shells such as Cygwin. For example, instead of running `imqcmd`, you must run `imqcmd.exe`.

To Create a JMS Host

The default JMS service configuration includes a JMS host, `default_JMS_host`. For most situations, this host is sufficient, so replacing it or creating additional JMS hosts is not often necessary and is a task for advanced users. Use the `create-jms-host` subcommand in remote `asadmin` mode to create an additional JMS host.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. Create the JMS host by using the "`create-jms-host`" subcommand:

```
asadmin> create-jms-host --mqhost hostName --mqport portNumber --mquser
adminUser --mqpassword adminPassword --target glassfishTarget --property
mqBrokerPropList --force trueFalse jms-host-name
```

- `--mqhost`: The host name of the Message Queue broker.
- `--mqport`: The port number of the Message Queue broker.
- `--mquser`: The username of the administrative user of the Message Queue broker.
- `--mqpassword`: The password of the administrative user of the Message Queue broker.
- `--target`: The Payara Server object for which the JMS host is being created. For details, see "`create-jms-host`".
- `--property`: A list of one or more Message Queue broker properties to configure the broker. The list is colon-separated (`:`) and has the form:

```
prop1Name=prop1Value:prop2Name=prop2Value:...
```

```
If a broker property name includes dots, preface the dots with two
backslashes (`\\`); for example, to include the `imq.system.max_count`
property, specify `imq\\.system\\.max_count` in the `--property` option. +
```

NOTE You can also set broker properties in the JMS service configuration. If you set the same broker property in both the JMS host and the JMS service configuration, the value specified in the JMS host is used.

- `--force`: Specifies whether the subcommand overwrites the existing JMS host of the same name. The default value is `false`.
- **jms-host-name**: The unique name of the JMS host.

Example 17-1 Creating a JMS Host

This example creates a JMS host named `MyNewHost`.

```
asadmin> create-jms-host --mqhost pigeon --mqport 7677 --mquser admin --mqpassword
admin MyNewHost
Jms Host MyNewHost created.
Command create-jms-host executed successfully.
```

To List JMS Hosts

Use the `list-jms-hosts` subcommand in remote `asadmin` mode to list the existing JMS hosts.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. List the JMS hosts by using the "`list-jms-hosts`" subcommand.

Example 17-2 Listing JMS Hosts

The following subcommand lists the existing JMS hosts.

```
asadmin> list-jms-hosts
default_JMS_host
MyNewHost
Command list-jmsdest executed successfully
```

To Update a JMS Host

Use the `set` subcommand in remote `asadmin` mode to update an existing JMS host.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. Use the "`get`" subcommand to list the current attribute values of the desired JMS host:

```
asadmin> get configs.config.config-name.jms-service.jms-host.jms-host-name.*
```

For information about JMS host attributes, see "`create-jms-host`(1)". . Use the "`set`" subcommand to modify a JMS host attribute:

```
asadmin> set configs.config.config-name.jms-service.jmshost.
jms-host-name.attribute-name=attribute-value
```

The attributes you can set are: * **`host`**: The host name of the Message Queue broker. * **`port`**: The port number of the Message Queue broker. * **`admin-user-name`**: The username of the administrative user of the Message Queue broker. * **`admin-`**

`password`: The password of the administrative user of the Message Queue broker. * **`property.`broker-property-name**: A Message Queue broker property. The property and the value assigned to it are used to configure the Message Queue broker.

+ If the broker property name includes dots, preface the dots with two backslashes (`\\`); for example, to include the `imq.system.max_count` property, specify `imq\\.system\\.max_count` in the `set` subcommand.

+

NOTE	You can also set broker properties in the JMS service configuration. If you set the same broker property in both the JMS host and the JMS service configuration, the value specified in the JMS host is used.

Example 17-3 Updating a JMS Host

This example changes the value of the `host` attribute of the JMS host `default_JMS_Host`. By default, this value is `localhost`.

```
asadmin> set configs.config.server-config.jms-service.jms-
host.default_JMS_host.host="server1.middleware.example.com"
```

To Delete a JMS Host

Use the `delete-jms-host` subcommand in remote `asadmin` mode to delete a JMS host from the JMS service. If you delete the only JMS host, the JMS service will not be able to start until you create a new JMS host.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. List the JMS hosts by using the "`list-jms-hosts`" subcommand.
3. Delete a JMS host by using the "`delete-jms-host`" subcommand.

Example 17-4 Deleting a JMS Host

This example deletes a JMS host named `MyNewHost`.

```
asadmin> delete-jms-host MyNewHost
Command delete-jms-host executed successfully.
```

Administering JMS Connection Factories and Destinations

The JMS API uses two kinds of administered objects. Connection factory objects allow an application to create other JMS objects programmatically. Destination objects serve as repositories for messages. How these objects are created is specific to each implementation of JMS. In Payara Server, JMS is implemented by performing the following tasks:

- Creating a connection factory
- Creating a destination, which requires creating a physical destination and a destination resource that refers to the physical destination

JMS applications use the Java Naming and Directory Interface (JNDI) API to access the connection factory and destination resources. A JMS application normally uses at least one connection factory and at least one destination. By studying the application or consulting with the application developer, you can determine what resources must be created. The order in which the resources are created does not matter.

The Jakarta EE standard specifies that certain default resources be made available to applications, and defines specific JNDI names for these default resources. Payara Server makes these names available through the use of logical JNDI names, which map Jakarta EE standard JNDI names to specific Payara Server resources. For JMS connection factory resources, the Jakarta EE standard name `java:comp/DefaultJMSConnectionFactory` is mapped to the `jms/__defaultConnectionFactory` resource.

Payara Server provides the following types of connection factory objects:

- `QueueConnectionFactory` objects, used for point-to-point communication
- `TopicConnectionFactory` objects, used for publish-subscribe communication
- `ConnectionFactory` objects, which can be used for both point-to-point and

publish-subscribe communications (recommended for new applications)

Payara Server provides the following types of destination objects:

- `Queue` objects, used for point-to-point communication
- `Topic` objects, used for publish-subscribe communication

The subcommands in this section can be used to administer both the connection factory resources and the destination resources. For information on JMS service support of connection pooling and failover, see connection-failover in the Payara Server High Availability section. For instructions on administering physical destinations, see administering-jms-physical-destinations.

To Create a Connection Factory or Destination Resource

For each JMS connection factory that you create, Payara Server creates a connector connection pool and connector resource. For each JMS destination that you create, Payara Server creates a connector admin object resource. If you delete a JMS resource, Payara Server automatically deletes the connector resources.

Use the `create-jms-resource` command in remote `asadmin` mode to create a JMS connection factory resource or a destination resource.

TIP

To specify the `addresslist` property (in the format `host:mqport,host2:mqport,host3:mqport`) for the `asadmin create-jms-resource` command, escape the : by using `\\`. For example, `host1\\:mqport,host2\\:mqport,host3\\:mpqport`.

To update a JMS connection factory, use the `set` subcommand for the underlying connector connection pool, See to-update-a-connector-connection-pool.

To update a destination, use the `set` subcommand for the admin object resource. See to-update-an-administered-object.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running

server.

2. Create a JMS resource by using the "`create-jms-resource`" command.
   Information about the properties for the subcommand is included in this help page.
3. If needed, restart the server.
   Some properties require server restart. See "Configuration Changes That Require Restart". If your server needs to be restarted, see to-restart-a-domain.

Example 17-5 Creating a JMS Connection Factory

This example creates a connection factory resource of type `jakarta.jms.ConnectionFactory` whose JNDI name is `jms/DurableConnectionFactory`. The `ClientId` property sets a client ID on the connection factory so that it can be used for durable subscriptions. The JNDI name for a JMS resource customarily includes the `jms/` naming subcontext.

```
asadmin> create-jms-resource --restype jakarta.jms.ConnectionFactory
--description "connection factory for durable subscriptions"
--property ClientId=MyID jms/DurableConnectionFactory
Command create-jms-resource executed successfully.
```

Example 17-6 Creating a JMS Destination

This example creates a destination resource whose JNDI name is `jms/MyQueue`.

```
asadmin> create-jms-resource --restype jakarta.jms.Queue
--property Name=PhysicalQueue jms/MyQueue
Command create-jms-resource executed successfully.
```

To List JMS Resources

Use the `list-jms-resources` subcommand in remote `asadmin` mode to list the existing connection factory and destination resources.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. List the existing JMS resources by using the "`list-jms-resources`" subcommand.

Example 17-7 Listing All JMS Resources

This example lists all the existing JMS connection factory and destination resources.

```
asadmin> list-jms-resources
jms/Queue
jms/ConnectionFactory
jms/DurableConnectionFactory
jms/Topic
Command list-jms-resources executed successfully
```

Example 17-8 Listing a JMS Resources of a Specific Type

This example lists the resources for the resource type `javax`.

```
asadmin> list-jms-resources --restype jakarta.jms.TopicConnectionFactory
jms/DurableTopicConnectionFactory
jms/TopicConnectionFactory
Command list-jms-resources executed successfully.
```

To Delete a Connection Factory or Destination Resource

Use the `delete-jms-resource` subcommand in remote `asadmin` mode to remove the specified connection factory or destination resource.

Before You Begin

Ensure that you remove all references to the specified JMS resource before running this subcommand.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. List the existing JMS resources by using the "`list-jms-resources`" subcommand.
3. Delete the JMS resource by using the "`delete-jms-resource`" subcommand.

Example 17-9 Deleting a JMS Resource

This example deletes the `jms/Queue` resource.

```
asadmin> delete-jms-resource jms/Queue
Command delete-jms-resource executed successfully
```

Administering JMS Physical Destinations

Messages are delivered for routing and delivery to consumers by using physical destinations in the JMS provider. A physical destination is identified and encapsulated by an administered object (such as a `Topic` or `Queue` destination resource) that an application component uses to specify the destination of messages it is producing and the source of messages it is consuming. For instructions on configuring a destination resource, see to-create-a-connection-factory-or-destination-resource.

If a message-driven bean is deployed and the physical destination it listens to does not exist, Payara Server automatically creates the physical destination and sets the value of the `maxNumActiveConsumers` property to `-1`. However, it is good practice to create the physical destination beforehand. The first time that an application accesses a destination resource, Message Queue automatically creates the physical destination specified by the Name property of the destination resource. This automatically created physical destination is temporary and expires after a period specified by a Message Queue configuration property, provided that there are no messages in it and no message producers or consumers connected to it.

To Create a JMS Physical Destination

For production purposes, always create physical destinations. During the development and testing phase, however, this step is not required. Use the `create-jmsdest` subcommand in remote `asadmin` mode to create a physical destination.

Because a physical destination is actually a Message Queue object rather than a server object, you use Message Queue broker commands to update properties.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. Create a JMS physical destination by using the "`create-jmsdest`" subcommand. Information about the properties for the subcommand is included in this help page.
3. If needed, restart the server. Some properties require server restart. See configuration-changes-that-require-restart. If your server needs to be restarted, see

to-restart-a-domain.

Example 17-10 Creating a JMS Physical Destination

This example creates a queue named `PhysicalQueue`.

```
asadmin> create-jmsdest --desttype queue --property
User=public:Password=public PhysicalQueue
Command create-jmsdest executed successfully.
```

To List JMS Physical Destinations

Use the `list-jmsdest` subcommand in remote `asadmin` mode to list the existing JMS physical destinations.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. List the existing JMS physical destinations by using the "`list-jmsdest`" subcommand.

Example 17-11 Listing JMS Physical Destinations

```
asadmin> list-jmsdest
PhysicalQueue queue {}
PhysicalTopic topic {}
Command list-jmsdest executed successfully.
```

This example lists the physical destinations for the default server instance.

To Purge Messages from a Physical Destination

Use the `flush-jmsdest` subcommand in remote `asadmin` mode to purge the messages from a physical destination in the specified target's JMS service configuration.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. Purge messages from the JMS physical destination by using the "`flush-jmsdest`" subcommand.

3. If needed, restart the server. Some properties require server restart. See configuration-changes-that-require-restart. If your server needs to be restarted, see to-restart-a-domain.

Example 17-12 Flushing Messages from a JMS Physical Destination

```
asadmin> flush-jmsdest --desttype queue PhysicalQueue
Command flush-jmsdest executed successfully
```

This example purges messages from the queue named `PhysicalQueue`.

To Delete a JMS Physical Destination

Use the `delete-jmsdest` subcommand in remote `asadmin` mode to remove the specified JMS physical destination.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. List the existing JMS physical destinations by using the "`list-jmsdest`" subcommand.
3. Delete the physical resource by using the "`delete-jmsdest`" subcommand.

Example 17-13 Deleting a Physical Destination

```
asadmin> delete-jmsdest --desttype queue PhysicalQueue
Command delete-jmsdest executed successfully
```

This example deletes the queue named `PhysicalQueue`.

Special Situations When Using the JMS Service

As mentioned earlier, Message Queue, through the built-in `jmsra` resource adapter, is tightly integrated with PayaraServer to provide JMS messaging managed through a rich set of `asadmin` subcommands and Administration Console pages to simplify JMS messaging administration tasks. In most instances, this tight integration is transparent and automatic, requiring no special effort on the part of an administrator. In certain special situations, though, an administrator must perform a task such a setting a Message Queue broker property or a Payara object attribute to enable or disable a

capability of the integration.

The topics in this section describe these situations.

- **Restarting an Embedded or Local Broker That Has Failed**: Because the JMS service, not Message Queue, manages the lifecycle of brokers acting as Embedded and Local JMS hosts, do not use the `imqbrokerd` Message Queue utility to start such a broker that has failed. Instead, restart the Payara instance that the broker is servicing.
- **Changing the Admin User Password for an Embedded or Local Broker**: Follow these steps to change the `admin` user password for an Embedded or Local broker:
  - Make sure the broker is running.
  - Use the `imqusermgr` Message Queue utility to change the password of the `admin` user.
  - Edit the configuration of the JMS host, changing the password of the `admin` user to the new password.
  - Restart the Payara instance that the broker is servicing.

    When changing the password for the brokers in a broker cluster, first perform steps 1 and 2 on each broker. Then, perform step 3. Finally, perform step 4 on each broker. Using SSL to Connect to an Oracle Internet Directory (OID) or Oracle
- **Virtual Directory (OVD) User Respository**: When using SSL to connect to an OID or OVD user repository, you must set the `imq.user_repository.ldap.ssl.socketfactory` Message Queue broker property to `com.sun.enterprise.security.auth.realm.ldap.CustomSocketFactory`.

Troubleshooting the JMS Service

If you encounter problems, consider the following:

- Use the "`jms-ping`" subcommand to confirm that the Message Queue broker is

running.

- View the Payara Server log file. For `server`, the DomainAdministrations Server (DAS), the log is available at domain-dir`/logs/server.log`; for other Payara instances, the log is available at as-install`/nodes/`node-name`/`instance-name`/logs/server.log`.
  If the log file indicates that a Message Queue broker acting as a Remote JMS host did not respond to a message, stop the broker and then restart it.
- View the broker log. For a broker associated with the Domain Administration Server (DAS), the log is available at domain-dir`/imq/instances/imqbroker/log/log.txt`; for brokers associated with other Payara instances, the log is available at as-install`/nodes/`node-name`/`instance-name`/imq/instances/`mq-instance-name`/log/log.txt`.
- For Remote type JMS hosts, be sure to start Message Queue brokers first, then Payara Server instances.
- If all Message Queue brokers are down, it can take up to 30 minutes for Payara Server to go down or up when you are using the default values in JMS. You can change the default values for this timeout. For example:

```
asadmin set domain1.jms-service.reconnect-interval-in-seconds=5
```

Using Activemq with Payara Server

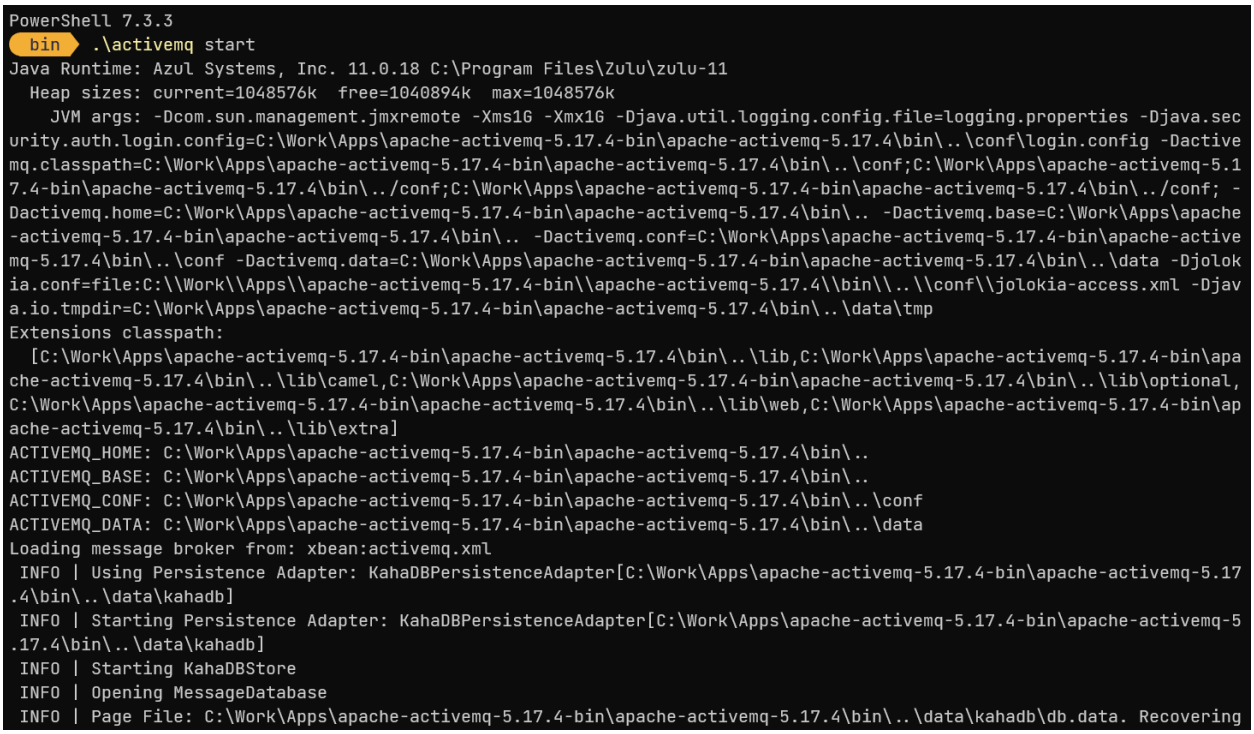
Payara Server comes with its own message queue broker: OpenMQ. However, ActiveMQ is also an option for a message queue broker. This section describes how to use ActiveMQ with Payara Server.

Installing Activemq

The latest version of ActiveMQ can be downloaded from here: ActiveMQ Downloads

Once downloaded, unzip the file and navigate to /bin folder and run the following command in order to start ActiveMQ:

```
activemq start
```



This is going to start the ActiveMQ broker at http://127.0.0.1:8161

After accessing the web console and logged in with the default credentials (`admin/admin`), you can create a new queue or topic by clicking on the Queues tab, inputting a name to Queue Name text box and clicking on "Create" button. For the purpose of this example we will create a queue named `TESTQ`.

Deploy the Activemq RAR to Payara Serve

In order to use ActiveMQ with Payara Server, you need to deploy the ActiveMQ RAR to Payara Server. You can download the ActiveMQ RAR from here: ActiveMQ RAR

The rar needs then to be deployed to Payara Server. This can be done by using the Payara Admin Console or by using the following asadmin command:

```
asadmin deploy --type rar activemq-rar-5.17.4.rar
```

Configuring the Activemq Connector

The first thing to do is to create a new JMS Resource Adapter Config. This can be done

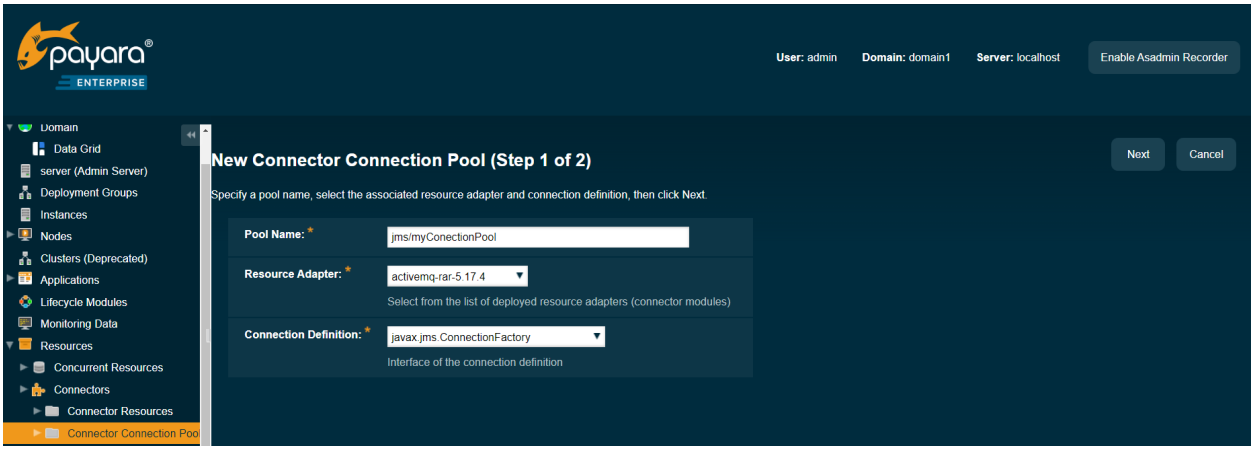
by using the Payara Admin Console or by using the following asadmin command:

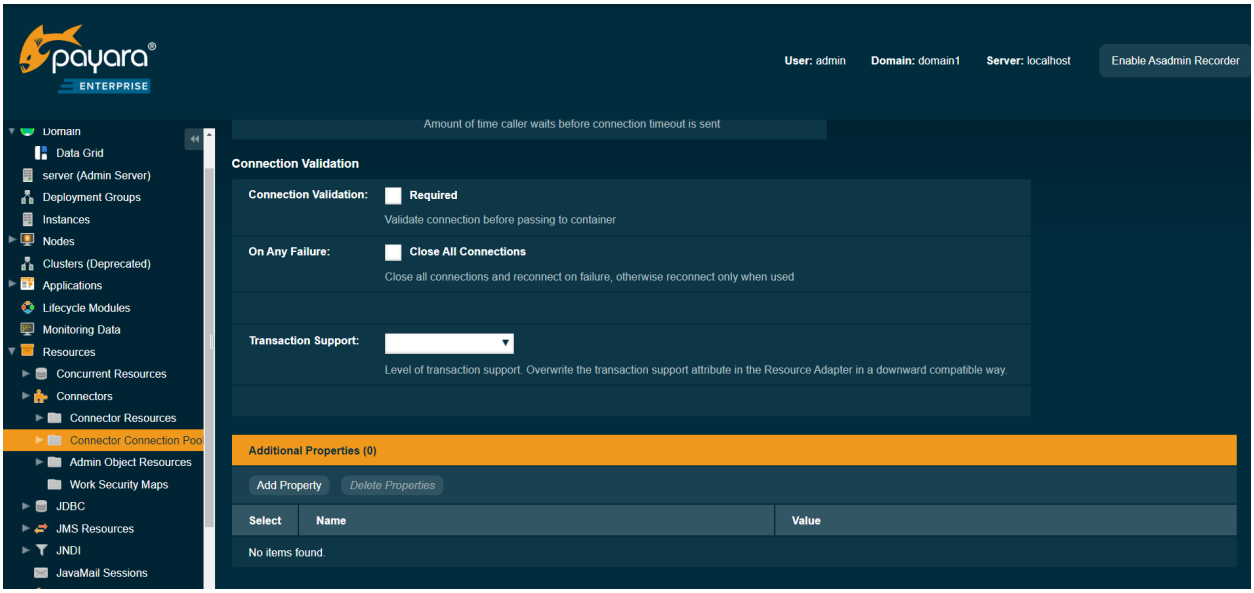
```
asadmin create-resource-adapter-config  --property ServerUrl
=tcp://127.0.0.1:61616:UserName='admin':Password='admin' activemq-rar-5.17.4
```

If you chose to use the Payara Admin Console, you will need to create a new JMS Resource Adapter Config by clicking on the "JMS Resource Adapter Configs" link under the "Resources" section of the left menu. The key properties for a default ActiveMQ configuration are:

Property	Value
ServerUrl	tcp://127.0.0.1:61616
UserName	admin
Password	admin

Now we need to create a new JMS Connector Connection Pool as shown in the following screenshots:

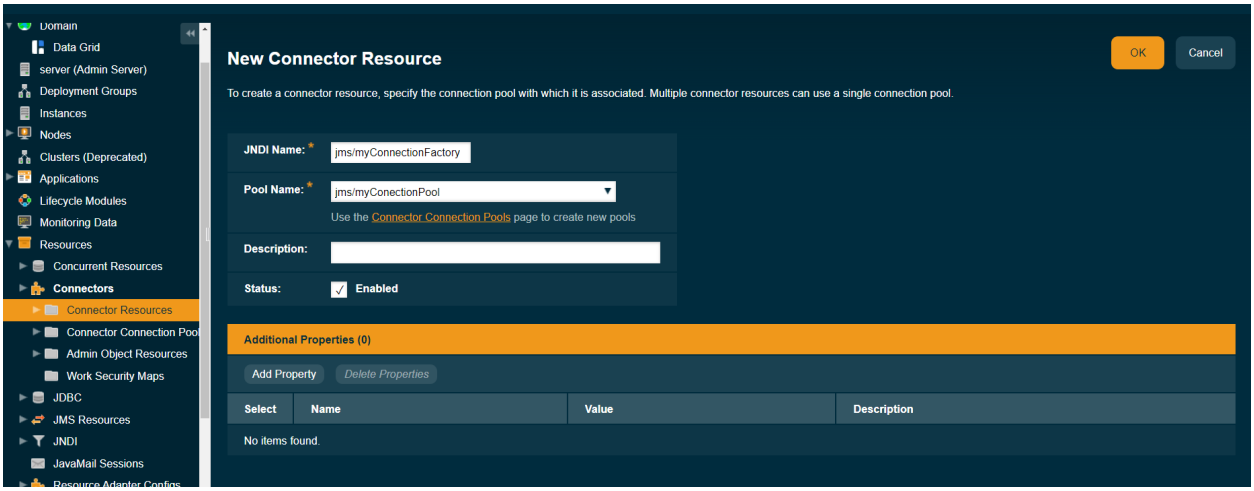




The Connector Connection Pool can also be created by using the following asadmin command:

```
asadmin create-connector-connection-pool --raname activemq-rar-5.17.4
--connectiondefinition javax.jms.ConnectionFactory --ping true
--isconnectvalidatereq true jms/myConnectionPool
```

Another configuration needed is the JNDI mapping for te JMS Connector Connection Pool. This can be done by using the Payara Admin Console following this screenshot:

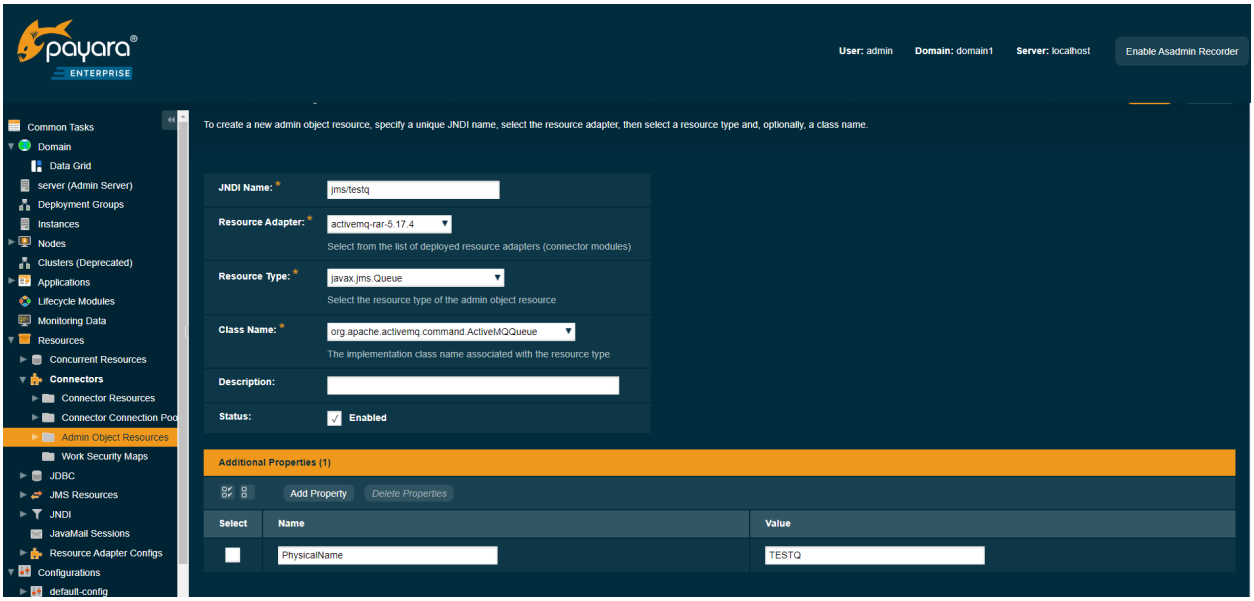


Alternatively you can use the following asadmin command:

```
asadmin create-connector-resource --poolname jms/myConnectionPool
```

```
jms/myConnectionFactory
```

Now we have the connection factory configured we can also create a JMS mapping to our Queue. This can be done by using the Payara Admin Console following this screenshot:



Again, if you prefer, you can use the following asadmin command to create the same mapping:

```
asadmin create-admin-object --raname activemq-rar-5.17.4 --restype javax.jms.Queue
--property PhysicalName=TESTQ jms/TESTQ
```

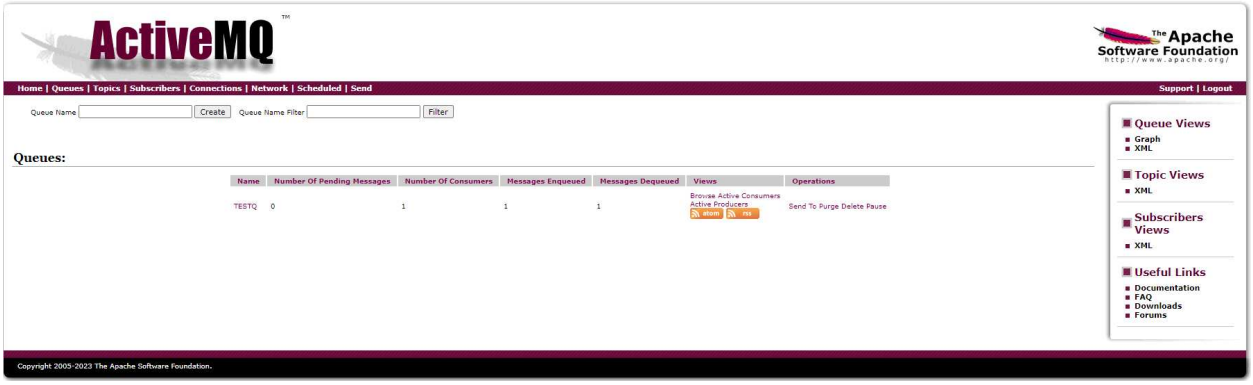
Testing the Activemq Configuration via MDB

Below shows the code for an MDB that listens on the queue configured in this blog within ActiveMQ. We can see that both the resource adapter, the physical name of the queue and the JNDI name are set as Activation Properties of the MDB.

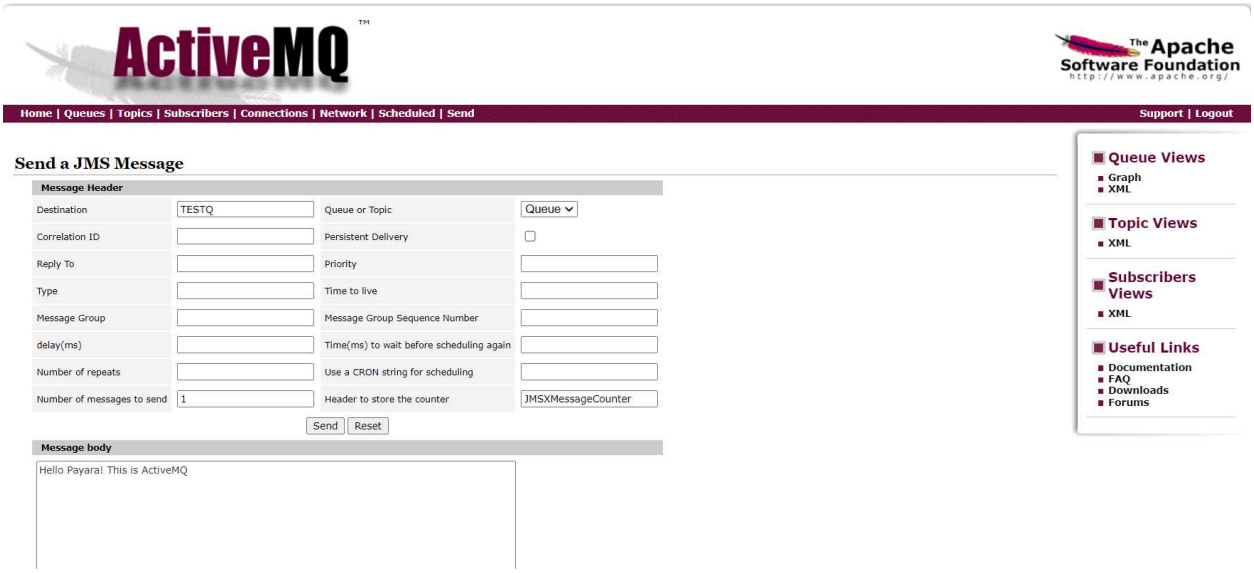
```
package fish.payara.support.activemqtest;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;

@MessageDriven(name = "testmdb", activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue =
"jms/TESTQ"),
```

```
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "destination", propertyValue =
"TESTQ"),
    @ActivationConfigProperty(propertyName = "resourceAdapter", propertyValue =
"activemq-rar-5.17.4")
})
public class NewMessageBean implements MessageListener {

    public NewMessageBean() {
    }

    @Override
    public void onMessage(Message message) {
        System.out.println("Got message " + message);
    }
}
```

If we build and deploy this MDB we will see in the Active MQ console that we have a single consumer on our queue.



We can test the MDB by sending a message using the ActiveMQ console.



Once you have sent the message we can check the server log, the message will be printed out:

```
imeMillis=1679392212239;_LevelValue=800;|
  Skipping registration of inhabitant for service reference [org.osgi.service.metatype.MetaTypeProvider] as the service
object could not be obtained.|#]

[#|2023-03-21T09:50:12.976+0000|INFO|Payara 5.48.1|javax.enterprise.system.core|_ThreadID=191;_ThreadName=Thread-16;_Tim
eMillis=1679392212976;_LevelValue=800;_MessageID=NCLS-CORE-00022;|
  Loading application __admingui done in 2,195 ms|#]

[#|2023-03-21T09:50:14.934+0000|INFO|Payara 5.48.1|javax.enterprise.resource.webcontainer.jsf.config|_ThreadID=191;_Thre
adName=Thread-16;_TimeMillis=1679392214934;_LevelValue=800;_MessageID=jsf.config.listener.version;|
  Initializing Mojarra 2.3.18.payara-p2 for context ''|#]

[#|2023-03-21T09:50:16.869+0000|INFO|Payara 5.48.1|javax.enterprise.web|_ThreadID=191;_ThreadName=Thread-16;_TimeMillis=
1679392216869;_LevelValue=800;_MessageID=AS-WEB-GLUE-00172;|
  Loading application [__admingui] at [/]|#]

[#|2023-03-21T10:02:34.776+0000|INFO|Payara 5.48.1|javax.enterprise.system.core.transaction.com.sun.jts.CosTransactions|
_ThreadID=235;_ThreadName=orb-thread-pool-1 (pool #1): worker-2;_TimeMillis=1679392954776;_LevelValue=800;|
  JTS5014: Recoverable JTS instance, serverId=[100]|#]

[#|2023-03-21T10:02:34.920+0000|INFO|Payara 5.48.1||_ThreadID=235;_ThreadName=orb-thread-pool-1 (pool #1): worker-2;_Tim
eMillis=1679392954920;_LevelValue=800;|
  Got message ActiveMQTextMessage {commandId = 7, responseRequired = false, messageId = ID:PCA-0034-63138-1679392134961-
4:1:1:1:1, originalDestination = null, originalTransactionId = null, producerId = ID:PCA-0034-63138-1679392134961-4:1:1:
1, destination = queue://TESTQ, transactionId = null, expiration = 0, timestamp = 1679392954532, arrival = 0, brokerInTi
me = 1679392954532, brokerOutTime = 1679392954544, correlationId = , replyTo = null, persistent = false, type = , priori
ty = 0, groupID = null, groupSequence = 0, targetConsumerId = null, compressed = false, userID = null, content = org.apa
che.activemq.util.ByteSequence@9514e5, marshalledProperties = null, dataStructure = null, redeliveryCounter = 0, size =
0, properties = null, readOnlyProperties = true, readOnlyBody = true, droppable = false, jmsXGroupFirstForConsumer = fal
se, text = Hello Payara! This is ActiveMQ!}|#]
```

Administering the Java Naming and Directory Interface (JNDI) Service

The Java Naming and Directory Interface (JNDI) API is used for accessing different kinds of naming and directory services. Jakarta EE components locate objects by invoking the JNDI lookup method.

Instructions for accomplishing the tasks in this chapter by using the Administration Console are contained in the Administration Console online help.

About JNDI

By making calls to the JNDI API, applications locate resources and other program objects. A resource is a program object that provides connections to systems, such as database servers and messaging systems. A JDBC resource is sometimes referred to as a data source. Each resource object is identified by a unique, people-friendly name, called the JNDI name. A resource object and its JNDI name are bound together by the naming and directory service, which is included with the Payara Server.

When a new name-object binding is entered into the JNDI, a new resource is created.

Jakarta EE Naming Environment

JNDI names are bound to their objects by the naming and directory service that is provided by a Jakarta EE server. Because Jakarta EE components access this service through the JNDI API, the object usually uses its JNDI name. For example, the JNDI name of the H2 database is `jdbc/_default`. At startup, the Payara Server reads information from the configuration file and automatically adds JNDI database names to the name space, one of which is `jdbc/_default`.

Jakarta EE application clients, enterprise beans, and web components must have access to a JNDI naming environment.

The application component's naming environment is the mechanism that allows customization of the application component's business logic during deployment or assembly. This environment allows you to customize the application component without needing to access or change the source code off the component. A Jakarta EE container implements the provides the environment to the application component instance as a JNDI naming context.

How the Naming Environment and the Container Work Together

The application component's environment is used as follows:

- The application component's business methods access the environment using the JNDI interfaces. In the deployment descriptor, the application component provider declares all the environment entries that the application component expects to be provided in its environment at runtime.
- The container provides an implementation of the JNDI naming context that stores the application component environment. The container also provides the tools that allow the deployer to create and manage the environment of each application component.
- A deployer uses the tools provided by the container to initialize the environment entries that are declared in the application component's deployment descriptor. The deployer sets and modifies the values of the environment entries.

- The container makes the JNDI context available to the application component instances at runtime. These instances use the JNDI interfaces to obtain the values of the environment entries.

Each application component defines its own set of environment entries. All instances of an application component within the same container share the same environment entries. Application component instances are not allowed to modify the environment at runtime.

Naming References and Binding Information

A resource reference is an element in a deployment descriptor that identifies the component's coded name for the resource. For example, `jdbc/SavingsAccountDB`. More specifically, the coded name references a connection factory for the resource.

The JNDI name of a resource and the resource reference name are not the same. This approach to naming requires that you map the two names before deployment, but it also decouples components from resources. Because of this decoupling, if at a later time the component needs to access a different resource, the name does not need to change. This flexibility makes it easier for you to assemble Jakarta EE applications from preexisting components.

The following table lists JNDI lookups and their associated resource references for the Jakarta EE resources used by the Payara Server.

Table 14. Table 18-1 JNDI Lookup Names and Their Associated References

JNDI Lookup Name	Associated Resource Reference
`java:comp/env`	Application environment entries
`java:comp/env/jdbc`	JDBC DataSource resource manager connection factories
`java:comp/env/ejb`	EJB References
`java:comp/UserTransaction`	UserTransaction references
`java:comp/env/mail`	JavaMail Session Connection Factories

JNDI Lookup Name	Associated Resource Reference
`java:comp/env/url`	URL Connection Factories
`java:comp/env/jms`	JMS Connection Factories and Destinations
`java:comp/ORB`	ORB instance shared across application components

Administering JNDI Resources

Within Payara Server, you can configure your environment for custom and external JNDI resources. A custom resource accesses a local JNDI repository; an external resource accesses an external JNDI repository. Both types of resources need user-specified factory class elements, JNDI name attributes, and so on.

- administering-custom-jndi-resources
- administering-external-jndi-resources

Administering Custom JNDI Resources

A custom resource specifies a custom server-wide resource object factory that implements the `javax.naming.spi.ObjectFactory` interface.

To Create a Custom JNDI Resource

Use the `create-custom-resource` subcommand in remote mode to create a custom resource.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create a custom resource by using the "`create-custom-resource`" subcommand.
   Information on properties for the subcommand is contained in this help page.
   Restart Payara Server. See to-restart-a-domain.

Example 18-1 Creating a Custom Resource

```
asadmin> create-custom-resource --restype topic --factoryclass com.imq.topic
sample_custom_resource
```

```
Command create-custom-resource executed successfully.
```

This example creates a custom resource named `sample-custom-resource`.

To List Custom JNDI Resources

Use the `list-custom-resources` subcommand in remote mode to list the existing custom resources.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the custom resources by using the "`list-custom-resources`" subcommand.

Example 18-2 Listing Custom Resources

```
asadmin> list-custom-resources
sample_custom_resource01
sample_custom_resource02
Command list-custom-resources executed successfully
```

This example lists the existing custom resources.

To Update a Custom JNDI Resource

1. List the custom resources by using the "`list-custom-resources`" subcommand.
2. Use the "`set`" subcommand to modify a custom JNDI resource.

Example 18-3 Updating a Custom JNDI Resource

```
asadmin> set server.resources.custom-resource.custom
/my-custom-resource.property.value=2010server.resources.custom-resource.custom
/my-custom-resource.property.value=2010
```

This example modifies a custom resource.

To Delete a Custom JNDI Resource

Use the `delete-custom-resource` subcommand in remote mode to delete a custom resource.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the custom resources by using the "`list-custom-resources`" subcommand.

3. Delete a custom resource by using the "`delete-custom-resource`" subcommand.

Example 18-4 Deleting a Custom Resource

```
asadmin> delete-custom-resource sample_custom_resource
Command delete-custom-resource executed successfully.
```

This example deletes a custom resource named `sample-custom-resource`.

Administering External JNDI Resources

Applications running on Payara Server often require access to resources stored in an external JNDI repository. For example, generic Java objects might be stored in an LDAP server according to the Java schema. External JNDI resource elements let you configure such external resource repositories.

To Register an External JNDI Resource

Use the `create-jndi-resource` subcommand in remote mode to register an external JNDI resource.

Before You Begin

The external JNDI factory must implement the `javax.naming.spi.InitialContextFactory` interface.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Register an external JNDI resource by using the "`create-jndi-resource`" subcommand.
   Information on properties for the subcommand is contained in this help page.
3. Restart Payara Server.
   See to-restart-a-domain.

Example 18-5 Registering an External JNDI Resource

```
asadmin> create-jndi-resource --jndilookupname sample_jndi
--restype queue --factoryclass sampleClass --description "this is a sample jndi
resource" sample_jndi_resource
Command create-jndi-resource executed successfully
```

In This example `sample_jndi_resource` is registered.

To List External JNDI Resources

Use the `list-jndi-resources` subcommand in remote mode to list all existing JNDI resources.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the existing JNDI resources by using the list-jndi-resources subcommand.

Example 18-6 Listing JNDI Resources

```
asadmin> list-jndi-resources
jndi_resource1
jndi_resource2
jndi_resource3
Command list-jndi-resources executed successfully
```

This example lists the JNDI resources.

To List External JNDI Entries

Use the `list-jndi-entries` subcommand in remote mode to browse and list the entries in the JNDI tree. You can either list all entries, or you can specify the JNDI context or subcontext to list specific entries.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the JNDI entries for a configuration by using the "`list-jndi-entries`" subcommand.

Example 18-7 Listing JNDI Entries

```
asadmin> list-jndi-entries
jndi_entry03
jndi_entry72
jndi_entry76
Command list-jndi-resources executed successfully
```

This example lists all the JNDI entries for the naming service.

To Update an External JNDI Resource

1. List the existing JNDI resources by using the list-jndi-resources subcommand.
2. Use the "`set`" subcommand to modify an external JNDI resource.

Example 18-8 Updating an External JNDI Resource

```
asadmin> set server.resources.external-jndi-resource.my-jndi-resource.
jndi-lookup-name=bar server.resources.external-jndi-resource.my-jndi-resource.jndi-
lookup-name=bar
```

This example modifies an external resource.

To Delete an External JNDI Resource

Use the `delete-jndi-resource` subcommand in remote mode to remove a JNDI resource.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Remove an external JNDI entry by using the "`delete-jndi-resource`" subcommand.

Example 18-9 Deleting an External JNDI Resource

```
asadmin> delete-jndi-resource jndi_resource2
Command delete-jndi-resource executed successfully.
```

This example deletes an external JNDI resource:

Example of Using an External JNDI Resource

```
<resources>
 <!-- external-jndi-resource element specifies how to access Jakarta EE
 - resources stored in an external JNDI repository. This example
 - illustrates how to access a java object stored in LDAP.
 - factory-class element specifies the JNDI InitialContext factory that
 - needs to be used to access the resource factory. property element
 - corresponds to the environment applicable to the external JNDI context
 - and jndi-lookup-name refers to the JNDI name to lookup to fetch the
 - designated (in this case the java) object.
 -->
  <external-jndi-resource jndi-name="test/myBean"
      jndi-lookup-name="cn=myBean"
      res-type="test.myBean"
      factory-class="com.sun.jndi.ldap.LdapCtxFactory">
    <property name="PROVIDER-URL" value="ldap://ldapserver:389/o=myObjects" />
```

```
    <property name="SECURITY_AUTHENTICATION" value="simple" />
    <property name="SECURITY_PRINCIPAL" value="cn=joeSmith, o=Engineering" />
    <property name="SECURITY_CREDENTIALS" value="changeit" />
  </external-jndi-resource>
</resources>
```

To Disable Payara Server v2 Vendor-Specific JNDI Names

The EJB 3.1 specification supported by Payara Server 5.0 defines portable EJB JNDI names. Because of this, there is less need to continue to use older vendor-specific JNDI names.

By default, Payara Server v2-specific JNDI names are applied automatically by Payara Server 5.0 for backward compatibility. However, this can lead to some ease-of-use issues. For example, deploying two different applications containing a Remote EJB component that exposes the same remote interface causes a conflict between the default JNDI names.

The default handling of v2-specific JNDI names in Payara Server can be managed with the `asadmin` command or with the `disable-nonportable-jndi-names` boolean property for the `ejb-container` element in `glassfish-ejb-jar.xml`.

Use the `asadmin` command or directly modify the `glassfish-ejb-jar.xml` file to set the `disable-nonportable-jndi-names` property.

- Using the `asadmin` command:

```
asadmin> set server.ejb-container.property.disable-nonportable-jndi-names="true"
```

- Directly modifying the `glassfish-ejb-jar.xml` file.

  1. Add the `disable-nonportable-jndi-names` property to the `ejb-container` element in `glassfish-ejb-jar.xml`.
  2. Set the value of the `disable-nonportable-jndi-names` boolean, as desired.
     `false` - Enables the automatic use of Payara Server v2-specific JNDI names. This is the default setting.
     `true` - Disables the automatic use of v2-specific JNDI names. In all cases, 5.0-

compatible JNDI names will be used.

3. Save the `glassfish-ejb-jar.xml` file and restart the Payara Server domain. This setting applies to all EJBs deployed to the server.

Administering Transactions

This chapter discusses how to manage the transaction service for the Payara Server environment by using the `asadmin` command-line utility.

Instructions for manually recovering transactions are also included.

Instructions for accomplishing the tasks in this chapter by using the Administration Console are contained in the Administration Console online help.

For more information about the Java Transaction API (JTA) and Java Transaction Service (JTS), see the following sites:

Jakarta Transactions API and Java Transaction Service.

You might also want to read Transactions.

About Transactions

A transaction is a series of discreet actions in an application that must all complete successfully. By enclosing one or more actions in an indivisible unit of work, a transaction ensures data integrity and consistency. If all actions do not complete, the changes are rolled back.

For example, to transfer funds from a checking account to a savings account, the following steps typically occur:

1. Check to see if the checking account has enough money to cover the transfer.
2. Debit the amount from the checking account.
3. Credit the amount to the savings account.
4. Record the transfer to the checking account log.

5. Record the transfer to the savings account log.

These steps together are considered a single transaction.

If all the steps complete successfully, the transaction is committed. If any step fails, all changes from the preceding steps are rolled back, and the checking account and savings account are returned to the states they were in before the transaction started. This type of event is called a `rollback`. A normal transaction ends in either a committed state or a rolled back state.

The following elements contribute to reliable transaction processing by implementing various APIs and functionalities:

- **Transaction Manager**. Provides the services and management functions required to support transaction demarcation, transactional resource management, synchronization, and transaction context propagation.
- **Payara Server**. Provides the infrastructure required to support the application runtime environment that includes transaction state management.
- **Resource Manager**. Through a resource adapter, the resource manager provides the application access to resources. The resource manager participates in distributed transactions by implementing a transaction resource interface used by the transaction manager to communicate transaction association, transaction completion, and recovery work. An example of such a resource manager is a relational database server.
- **Resource Adapter**. A system-level software library is used by Payara Server or a client to connect to a resource manager. A resource adapter is typically specific to a resource manager. The resource adapter is available as a library and is used within the address space of the client using it. An example of such a resource adapter is a Java Database Connectivity (JDBC) driver. For information on supported JDBC drivers, see configuration-specifics-for-jdbc-drivers.
- **Transactional User Application**. In the Payara Server environment, the transactional user application uses Java Naming and Directory Interface (JNDI) to look up

transactional data sources and, optionally, the user transaction). The application might use declarative transaction attribute settings for enterprise beans, or explicit programmatic transaction demarcation. For more information, see the-transaction-manager-the-transaction-synchronization-registry-and-usertransaction in the Payara Server Application Development section.

Transaction Resource Managers

There are three types of transaction resource managers:

- **Databases** - Use of transactions prevents databases from being left in inconsistent states due to incomplete updates. For information about JDBC transaction isolation levels, see using-jdbc-transaction-isolation-levels in the Payara Server Application Development section.
  The Payara Server supports a variety of JDBC XA drivers. For configurations of supported and other drivers, see configuration-specifics-for-jdbc-drivers.
- **Java Message Service (JMS) Providers** - Use of transactions ensures that messages are reliably delivered. The Payara Server is integrated with Open Message Queue, a fully capable JMS provider. For more information about transactions and the JMS API, see administering-the-java-message-service-jms.
- **J2EE Connector Architecture (CA) components** - Use of transactions prevents legacy EIS systems from being left in inconsistent states due to incomplete updates. For more information about connectors, see administering-eis-connectivity.

Transaction Scope

A local transaction involves only one non-XA resource and requires that all participating application components execute within one process. Local transaction optimization is specific to the resource manager and is transparent to the Jakarta EE application.

In the Payara Server, a JDBC resource is non-XA if it meets either of the following criteria:

- In the JDBC connection pool configuration, the DataSource class does not implement the javax.sql.XADataSource interface.

- The Resource Type setting is not set to javax.sql.XADataSource.

A transaction remains local if the following conditions remain true:

- One and only one non-XA resource is used. If any additional non-XA resource is used, the transaction is aborted, because the transaction manager must use XA protocol to commit two or more resources.
- No transaction importing or exporting occurs.

Transactions that involve multiple resources or multiple participant processes are distributed or global transactions. A global transaction can involve one non-XA resource if last agent optimization is enabled. Otherwise, all resources must be XA. The `use-last-agent-optimization` property is set to `true` by default. For details about how to set this property, see configuring-the-transaction-service.

If only one XA resource is used in a transaction, one-phase commit occurs, otherwise the transaction is coordinated with a two-phase commit protocol.

A two-phase commit protocol between the transaction manager and all the resources enlisted for a transaction ensures that either all the resource managers commit the transaction or they all abort. When the application requests the commitment of a transaction, the transaction manager issues a `PREPARE_TO_COMMIT` request to all the resource managers involved. Each of these resources can in turn send a reply indicating whether it is ready for commit (`PREPARED`) or not (`NO`). Only when all the resource managers are ready for a commit does the transaction manager issue a commit request (`COMMIT`) to all the resource managers. Otherwise, the transaction manager issues a rollback request (`ABORT`) and the transaction is rolled back.

Configuring the Transaction Service

You can configure the transaction service in the Payara Server in the following ways:

- To configure the transaction service using the Administration Console, open the Transaction Service component under the relevant configuration. For details, click the Help button in the Administration Console.

- To configure the transaction service, use the "`set`" subcommand to set the following attributes.

  The following examples show the `server-config` configuration, but values for any configuration can be set. For example, if you create a cluster named `cluster1` and a configuration named `cluster1-config` is automatically created for it, you can use `cluster1-config` in the `set` subcommand to get the transaction service settings for that cluster.

```
server-config.transaction-service.automatic-recovery = false
server-config.transaction-service.heuristic-decision = rollback
server-config.transaction-service.keypoint-interval = 2048
server-config.transaction-service.retry-timeout-in-seconds = 600
server-config.transaction-service.timeout-in-seconds = 0
server-config.transaction-service.tx-log-dir = domain-dir/logs
```

  You can also set these properties:

```
server-config.transaction-service.property.oracle-xa-recovery-workaround = true
server-config.transaction-service.property.sybase-xa-recovery-workaround = false
server-config.transaction-service.property.disable-distributed-transaction-logging
= false
server-config.transaction-service.property.xaresource-txn-timeout = 0
server-config.transaction-service.property.pending-txn-cleanup-interval = -1
server-config.transaction-service.property.use-last-agent-optimization = true
server-config.transaction-service.property.delegated-recovery = false
server-config.transaction-service.property.wait-time-before-recovery-insec = 60
server-config.transaction-service.property.purge-cancelled-transactions-after = 0
server-config.transaction-service.property.commit-one-phase-during-recovery = false
server-config.transaction-service.property.add-wait-point-during-recovery = 0
server-config.transaction-service.property.db-logging-resource = jdbc/TxnDS
server-config.transaction-service.property.xa-servername = myserver
server-config.transaction-service.property.recover-txn-delay = -1
```

Default property values are shown where they exist. For `db-logging-resource` and `xa-servername`, typical values are shown. Values that are not self-explanatory are as follows:

- The `xaresource-txn-timeout` default of `0` means there is no timeout. The timeout value is expressed in seconds.
- The `pending-txn-cleanup-interval` default of `-1` means the periodic recovery thread doesn't run. The interval value is expressed in seconds.

- The `purge-cancelled-transactions-after` default of `0` means cancelled transactions are not purged. The value is expressed as the number of cancellations between purge attempts.
- The `add-wait-point-during-recovery` property does not have a default value. If this property is unset, recovery does not wait. The wait value is expressed in seconds.
- The `db-logging-resource` property does not have a default value. It is unset by default. However, if you set `db-logging-resource` to an empty value, the value used is `jdbc/TxnDS`.
- The `xa-servername` property does not have a default value. Use this property to override server names that can cause errors.
- The `recover-txn-delay` default of `-1` means that recovery will be attempted upon start of the server if `automatic-recovery` is enabled. Use this property to delay the initial execution of the recovery thread. The delay value is expressed in seconds.

You can use the "`get`" subcommand to list all the transaction service attributes and the properties that have been set.
Changing `keypoint-interval`, `retry-timeout-in-seconds`, or `timeout-in-seconds` does not require a server restart. Changing other attributes or properties requires a server restart. * You can also set the following system properties:

+

```
ALLOW_MULTIPLE_ENLISTS_DELISTS=false
JTA_RESOURCE_TABLE_MAX_ENTRIES=8192
JTA_RESOURCE_TABLE_DEFAULT_LOAD_FACTOR=0.75f
```

The `JTA_RESOURCE_TABLE_DEFAULT_LOAD_FACTOR` default is the default `Map` resizing value.

Managing the Transaction Service for Rollbacks

You can roll back a single transaction by using the `asadmin` subcommands described in this section. To do so, the transaction service must be stopped (and later restarted),

allowing you to see the active transactions and correctly identify the one that needs to be rolled back.

To Stop the Transaction Service

Use the `freeze-transaction-service` subcommand in remote mode to stop the transaction service. When the transaction service is stopped, all in-flight transactions are immediately suspended. You must stop the transaction service before rolling back any in-flight transactions.

Running this subcommand on a stopped transaction subsystem has no effect. The transaction service remains suspended until you restart it by using the `unfreeze-transaction-service` subcommand.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Stop the transaction service by using the "`freeze-transaction-service`" subcommand.

Example 19-1 Stopping the Transaction Service

This example stops the transaction service.

```
asadmin> freeze-transaction-service --target instance1
Command freeze-transaction-service executed successfully
```

To Roll Back a Transaction

In some situations, you might want to roll back a particular transaction. Before you can roll back a transaction, you must first stop the transaction service so that transaction operations are suspended. Use the `rollback-transaction` subcommand in remote mode to roll back a specific transaction.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Enable monitoring using the `set` subcommand. For example:

```
asadmin> set cluster1-config.monitoring-service.module-monitoring-
levels.transaction-service=HIGH
```

3. Use the `freeze-transaction-service` subcommand to halt in-process transactions. See to-stop-the-transaction-service.
4. Identify the ID of the transaction you want to roll back.
   To see a list of IDs of active transactions, use the `get` subcommand with the `--monitor` option to get the monitoring data for the `activeids` statistic. See transaction-service-statistics. For example:

```
asadmin> get --monitor instance1.server.transaction-service.activeids-current
```

5. Roll back the transaction by using the "`rollback-transaction`" subcommand.
   The transaction is not rolled back at the time of this command's execution, but only marked for rollback. The transaction is rolled back when it is completed.

Example 19-2 Rolling Back a Transaction

This example rolls back the transaction with transaction ID `0000000000000001_00`.

```
asadmin> rollback-transaction --target instance1 0000000000000001_00
Command rollback-transaction executed successfully
```

To Restart the Transaction Service

Use the `unfreeze-transaction-service` subcommand in remote mote to resume all the suspended in-flight transactions. Run this subcommand to restart the transaction service after it has been frozen.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Restart the suspended transaction service by using the "`unfreeze-transaction-service`" subcommand.

Example 19-3 Restarting the Transaction Service

This example restarts the transaction service after it has been frozen.

```
asadmin> unfreeze-transaction-service --target instance1
Command unfreeze-transaction-service executed successfully
```

Determining Local Transaction Completion at Shutdown

When you shut down a Payara Server instance, all database connections are closed. When an Oracle JDBC driver-based database connection is closed in the middle of a non-XA transaction, all pending changes are committed. Other databases usually roll back pending changes when a connection is closed without being explicitly committed. To determine the exact behavior for your database, refer to the documentation from your JDBC driver vendor.

To explicitly specify whether Payara Server commits or rolls back non-XA transactions at server shutdown, set the `com.sun.enterprise.in-progress-local-transaction.completion-mode` JVM option to either `commit` or `rollback` using the "`create-jvm-options`" subcommand. For example:

```
asadmin> create-jvm-options -Dcom.sun.enterprise.in-progress-local-
transaction.completion-mode=rollback
```

Recovering Transactions

There are some situations where the commit or rollback operations might be interrupted, typically because the server crashed or a resource manager crashed. Crash situations can leave some transactions stranded between steps. Payara Server is designed to recover from these failures. If the failed transaction spans multiple servers, the server that started the transaction can contact the other servers to get the outcome of the transaction. If the other servers are unreachable, the transaction uses heuristic decision information to determine the outcome.

Automatic Transaction Recovery

Payara Server can perform automatic recovery in these ways:

- Pending transactions are completed upon server startup if `automatic-recovery` is set to `true`.
- Periodic automatic recovery is performed by a background thread if the `pending-txn-cleanup-interval` property is set to a positive value.

Changing these settings requires a server restart. For more information about how to

change these settings, see configuring-the-transaction-service. If commit fails during recovery, a message is written to the server log.

To Manually Recover Transactions

Use the `recover-transactions` subcommand in remote mode to manually recover transactions that were pending when a resource or a server instance failed.

For a standalone server, do not use manual transaction recovery to recover transactions after a server failure. For a standalone server, manual transaction recovery can recover transactions only when a resource fails, but the server is still running. If a standalone server fails, only the full startup recovery process can recover transactions that were pending when the server failed.

For an installation of multiple server instances, you can use manual transaction recovery from a surviving server instance to recover transactions after a server failure. For manual transaction recovery to work properly, transaction logs must be stored on a shared file system that is accessible to all server instances. See transaction-logging.

When you execute `recover-transactions` in non-delegated mode, you can recover transactions that didn't complete two-phase commit because of a resource crash. To use manual transaction recovery in this way, the following conditions must be met:

- The `recover-transactions` command should be executed after the resource is restarted.
- Connection validation should be enabled so the connection pool is refreshed when the resource is accessed after the recovery.

If commit fails during recovery, a message is written to the server log.

NOTE A JMS resource crash is handled the same way as any other resource. You can list in-doubt Open Message Queue transactions using the `imqcmd list txn` subcommand.

1. Ensure that the server is running. Remote subcommands require a running server.

2. Manually recover transactions by using the "`recover-transactions`" subcommand.

Example 19-4 Manually Recovering Transactions

This example performs manual recovery of transactions on `instance1`, saving them to `instance2`.

```
asadmin recover-transactions --target instance2 instance1
Transaction recovered.
```

Distributed Transaction Recovery

To enable cluster-wide automatic recovery, you must first facilitate storing of transaction logs in a shared file system. See transaction-logging.

Next, you must set the transaction service's `delegated-recovery` property to `true` (the default is `false`). For information about setting `tx-log-dir` and `delegated-recovery`, see configuring-the-transaction-service.

Recovery Workarounds and Limitations

The Payara Server provides workarounds for some known issues with transaction recovery implementations.

NOTE	These workarounds do not imply support for any particular JDBC driver.

General Recovery Limitations

The following general limitations apply to transaction recovery:

- Recovery succeeds if there are no exceptions during the process. This is independent of the number of transactions that need to be recovered.
- Only transactions that did not complete the two-phase commit can be recovered (one of the XA resources failed or Payara Server crashed after resources were prepared).
- Manual transaction recovery cannot recover transactions after a server crash on a standalone server instance. Manual operations are intended for cases when a resource dies unexpectedly while the server is running. In case of a server crash, only

startup recovery can recover in-doubt transactions.

- It is not possible to list transaction IDs for in-doubt transactions.
- Delegated transaction recovery (by a different server instance in a cluster) is not possible if the failed instance used an `EMBEDDED` Message Queue broker, or if it used a `LOCAL` or `REMOTE` Message Queue broker and the broker also failed. In this case, only automatic recovery on server instance restart is possible. This is because for conventional Message Queue clustering, state information in a failed broker is not available until the broker restarts.

Oracle Setup for Transaction Recovery

You must configure the following `grant` statements in your Oracle database to set up transaction recovery:

```
grant select on SYS.DBA_PENDING_TRANSACTIONS to user;
grant execute on SYS.DBMS_SYSTEM to user;
grant select on SYS.PENDING_TRANS$ to user;
grant select on SYS.DBA_2PC_NEIGHBORS to user;
grant execute on SYS.DBMS_XA to user;
grant select on SYS.DBA_2PC_PENDING to user;
```

The user is the database administrator. On some versions of the Oracle driver the last `grant execute` fails. You can ignore this.

Oracle Thin Driver

In the Oracle thin driver, the `XAResource.recover` method repeatedly returns the same set of in-doubt Xids regardless of the input flag. According to the XA specifications, the Transaction Manager initially calls this method with TMSTARTSCAN and then with TMNOFLAGS repeatedly until no Xids are returned. The `XAResource.commit` method also has some issues.

To disable the Payara Server workaround, set the `oracle-xa-recovery-workaround` property value to `false`. For details about how to set this property, see configuring-the-transaction-service. This workaround is used unless explicitly disabled.

Delegated Recovery after Server Crash Doesn't Work on Mysql

The MySQL database supports XA transaction recovery only when the database crashes. When a Payara Server instance crashes, MySQL rolls back prepared transactions.

Call to `XATeminator.recover()` During `ResourceAdapter.start()` Hangs If Automatic Recovery Is Enabled

Calls to `XATerminator.recover()` from the `ResourceAdapter.start()` method never return because Payara Server deadlocks. This only occurs when automatic recovery is enabled.

It is not advisable to do transactional activities, such as starting a transaction or calling `XATerminator.recover()`, during `ResourceAdapter.start()`. For more information, see `http://markmail.org/message/ogc7qndhaywfkdrp#query:+page:1+mid:kyyzpcexusbnv7ri+state:results`.

Transaction Logging

The transaction service writes transactional activity into transaction logs so that transactions can be recovered. You can control transaction logging in these ways:

- Set the location of the transaction log files in one of these ways:
  - Set the Payara Server's `log-root` setting to a shared file system base directory and set the transaction service's `tx-log-dir` attribute to a relative path.
  - Set `tx-log-dir` to an absolute path to a shared file system directory, in which case `log-root` is ignored for transaction logs.
  - Set a system property called `TX-LOG-DIR` to a shared file system directory. For example:

```
asadmin> create-system-properties --target server TX-LOG-DIR=/inst1/logs
```

For information about setting `log-root` and other general logging settings, see administering-the-logging-service.

- Turn off transaction logging by setting the `disable-distributed-transaction-logging` property to `true` and the `automatic-recovery` attribute to `false`. Do this only if performance is more important than transaction recovery.

NOTE	All instances in a cluster must be owned by the same user (`uid`), and read/write permissions for that user must be set on the transaction log directories.

Transaction logs should be stored in a high-availability network file system (NFS) to avoid a single point of failure.

To Store Transaction Logs in a Database

For multi-core machines, logging transactions to a database may be more efficient. Transaction logging is designed to work with any JDBC-compliant database. For databases with which transaction logging has been tested, see the Payara Server Release Notes.

1. Create a JDBC connection Pool. To use non-transactional connections to insert log records, you can either set the `non-transactional-connections` attribute to `true` in this step, or you can perform step 5 later.
2. Create a JDBC resource that uses the connection pool and note the JNDI name of the JDBC resource.
3. Automatic table creation for the transaction logs is done by default. However, if you would prefer to create the table manually, name it `txn_log_table` with the following schema:

Column Name	JDBC Type
`LOCALTID`	`VARCHAR`
`INSTANCENAME`	`VARCHAR`
`SERVERNAME`	`VARCHAR(n)`
`GTRID`	`VARBINARY`

The size of the `SERVERNAME` column should be at least the length of the Payara

Server host name plus 10 characters.

The size of the `GTRID` column should be at least 64 bytes.

4. Add the `db-logging-resource` property to the transaction service. For example:

```
asadmin set server-config.transaction-service.property.db-logging-resource
="jdbc/TxnDS"
```

   The property's value should be the JNDI name of the JDBC resource configured previously.

5. If you didn't set the `non-transactional-connections` attribute to `true` in step 1 and you want to use non-transactional connections to insert log records, use the following `asadmin create-jvm-options` command to reference an existingtransactional resource but use non-transactional connections for the `INSERT` statements:

```
asadmin create-jvm-options -Dcom.sun.jts.dblogging.use.nontx.connection.for.add
=true
```

6. To disable file synchronization, use the following `asadmin create-jvm-options` command:

```
asadmin create-jvm-options -Dcom.sun.appserv.transaction.nofdsync
```

7. Restart the server.

Next Steps

To define the SQL used by the transaction manager when it is storing its transaction logs in the database, use the following flags:

`-Dcom.sun.jts.dblogging.insertquery=sql` statement

`-Dcom.sun.jts.dblogging.deletequery=sql` statement

`-Dcom.sun.jts.dblogging.selectquery=sql` statement

`-Dcom.sun.jts.dblogging.selectservernamequery=sql` statement

The default statements are as follows:

```
-Dcom.sun.jts.dblogging.insertquery=insert into txn_log_table
values ( ?, ?, ?, ? )
```

```
-Dcom.sun.jts.dblogging.deletequery=delete from txn_log_table
where localtid = ? and servername = ?
```

```
-Dcom.sun.jts.dblogging.selectquery=select * from txn_log_table
where servername = ?
```

```
-Dcom.sun.jts.dblogging.selectservernamequery=select distinct
servername from txn_log_table where instancename = ?
```

To set one of these flags using the `asadmin create-jvm-options` command, you must quote the statement. For example:

```
create-jvm-options '-Dcom.sun.jts.dblogging.deletequery=delete
from txn_log_table where gtrid = ?'
```

You can also set JVM options in the Administration Console. Select the JVM Settings component under the relevant configuration. These flags and their statements must also be quoted in the Administration Console. For example:

```
'-Dcom.sun.jts.dblogging.deletequery=delete from txn_log_table
where gtrid = ?'
```

Administering Web Applications

This chapter explains how to administer web applications in the Payara Server environment.

Instructions for accomplishing some of these tasks by using the Administration Console are contained in the Administration Console online help.

Invoking a Servlet by Alternate Means

You can call a servlet deployed to Payara Server by using a URL in a browser or embedded as a link in an HTML or JSP file. The format of a servlet invocation URL is as

follows:

```
http://server:port/context-root/servlet-mapping?name=value
```

Table 15. Table 6-1 URL Fields for Servlets Within an Application

URL element	Description
`server:port`	The IP address (or host name) and optional port number. To access the default web module for a virtual server, specify only this URL section. You do not need to specify the context-root or servlet-name unless you also wish to specify name-value parameters.
context-root	For an application, the context root is defined in the `context-root` element of the `application.xml`, `sun-application.xml`, or `payara-web.xml`/`sun-web.xml` (deprecated) file. For an individually deployed web module, the context root is specified during deployment. For both applications and individually deployed web modules, the default context root is the name of the WAR file minus the `.war` suffix.
servlet-mapping	The `servlet-mapping` as configured in the `web.xml` file.
`?`name`=`value`...`	Optional request parameters.

In this example, `localhost` is the host name, `MortPages` is the context root, and `calcMortgage` is the servlet mapping.

Example 6-1 Invoking a Servlet with a URL

```
http://localhost:8080/MortPages/calcMortgage?rate=8.0&per=360&bal=180000
```

To invoke a servlet from within a JSP file, you can use a relative path. For example:

Example 6-2 Invoking a Servlet from within a JSP File

```
<jsp:forward page="TestServlet"/><jsp:include page="TestServlet"/>
```

Changing Log Output for a Servlet

`ServletContext.log` messages are sent to the server log. By default, the `System.out` and `System.err` output of servlets are sent to the server log. During

startup, server log messages are echoed to the `System.err` output. Also by default, there is no Windows-only console for the `System.err` output.

You can change these defaults using the Administration Console Write to System Log box. If this box is checked, `System.out` output is sent to the server log. If it is unchecked, `System.out` output is sent to the system default location only.

Defining Global Features for Web Applications

You can use the `default-web.xml` file to define features such as filters and security constraints that apply to all web applications.

For example, directory listings are disabled by default for added security. To enable directory listings in your domain's `default-web.xml` file, search for the definition of the servlet whose `servlet-name` is equal to `default`, and set the value of the `init-param` named `listings` to `true`. Then restart the server.

```
<init-param>
   <param-name>listings</param-name>
   <param-value>true</param-value>
</init-param>
```

If `listings` is set to `true`, you can also determine how directory listings are sorted. Set the value of the `init-param` named `sortedBy` to `NAME`, `SIZE`, or `LAST_MODIFIED`. Then restart the server.

```
<init-param>
   <param-name>sortedBy</param-name>
   <param-value>LAST_MODIFIED</param-value>
</init-param>
```

The `mime-mapping` elements in `default-web.xml` are global and inherited by all web applications. You can override these mappings or define your own using `mime-mapping` elements in your web application's `web.xml` file. For more information about `mime-mapping` elements, see the Servlet specification.

You can use the Administration Console to edit the `default-web.xml` file, or edit the file directly using the following steps.

To Use the `default-web.xml` File

1. Place the JAR file for the filter, security constraint, or other feature in the domain-dir`/lib` directory.
2. Edit the domain-dir`/config/default-web.xml` file to refer to the JAR file.
3. To apply your changes, restart Payara Server. See to-restart-a-domain.

Redirecting a URL

You can specify that a request for an old URL be treated as a request for a new URL. This is called redirecting a URL.

To specify a redirected URL for a virtual server, use the `redirect_n` property, where n is a positive integer that allows specification of more than one. Each of these `redirect_n` properties is inherited by all web applications deployed on the virtual server.

The value of each `redirect_n` property has two components which can be specified in any order:

- The first component, `from`, specifies the prefix of the requested URI to match.
- The second component, `url-prefix`, specifies the new URL prefix to return to the client. The from prefix is replaced by this URL prefix.

This example redirects `from dummy` to `etude`:

Example 6-3 Redirecting a URL

```
<property name="redirect_1" value="from=/dummy url-prefix=http://etude"/>
```

Administering `mod_jk`

The Apache Tomcat Connector `mod_jk` can be used to connect the web container with web servers such as Apache HTTP Server. By using `mod_jk`, which comes with Payara Server, you can front Payara Server with Apache HTTP Server.

You can also use `mod_jk` directly at the JSP/servlet engine for load balancing. For more information about configuring `mod_jk` and Apache HTTP Server for load balancing with

Payara Server refer to configuring-http-load-balancing in the Payara Server High Availability section.

To Enable `mod_jk`

You can front Payara Server with Apache HTTP Server by enabling the `mod_jk` protocol for one of Payara Server's network listeners, as described in this procedure. A typical use for `mod_jk` would be to have Apache HTTP Server handle requests for static resources, while having requests for dynamic resources, such as servlets and JavaServer Pages (JSPs), forwarded to, and handled by the Payara Server back-end instance.

When you use the `jk-enabled` attribute of the network listener, you do not need to copy any additional JAR files into the `/lib` directory. You can also create JK connectors under different virtual servers by using the network listener attribute `jk-enabled`.

1. Install Apache HTTP Server and `mod_jk`.
    - For information on installing Apache HTTP Server, see `https://httpd.apache.org/docs/2.4/install.html`.
    - For information on installing `mod_jk`, see `http://tomcat.apache.org/connectors-doc/webserver_howto/apache.html`.
2. Configure the following files:
    - `apache2/conf/httpd.conf`, the main Apache configuration file
    - `apache2/conf/workers.properties` example-6-4 and example-6-5 provide examples of configuring these two files.
3. Start Apache HTTP Server (`httpd`).
4. Start Payara Server with at least one web application deployed.
   In order for the `mod_jk`-enabled network listener to start listening for requests, the web container must be started. Normally, this is achieved by deploying a web application.

5. Create a jk-enabled network listener by using the "`create-network-listener`" subcommand.

```
asadmin> create-network-listener --protocol http-listener-1 \
--listenerport 8009 --jkenabled true jk-connector
```

6. If you are using the `glassfish-jk.properties` file to use non-default values of attributes described at `http://tomcat.apache.org/tomcat-5.5-doc/config/ajp.html`), set the `jk-configuration-file` property of the network listener to the fully-qualified file name of the `glassfish-jk.properties` file.

```
asadmin> set server-config.network-config.network-listeners.network-listener.\
jk-connector.jk-configuration-file=domain-dir/config/glassfish-jk.properties
```

7. If you expect to need more than five threads for the listener, increase the maximum threads in the `http-thread-pool` pool:

```
asadmin> set configs.config.server-config.thread-pools.thread-pool.\
http-thread-pool.max-thread-pool-size=value
```

8. To apply your changes, restart Payara Server. See to-restart-a-domain.

Example 6-4 `httpd.conf` File for `Mod_jk`

This example shows an `httpd.conf` file that is set for `mod_jk`. In this example, `mod_jk` used as a simple pass-through.

```
LoadModule jk_module /usr/lib/httpd/modules/mod_jk.so
JkWorkersFile /etc/httpd/conf/worker.properties
# Where to put jk logs
JkLogFile /var/log/httpd/mod_jk.log
# Set the jk log level [debug/error/info]
JkLogLevel debug
# Select the log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
# JkOptions indicate to send SSL KEY SIZE,
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories
# JkRequestLogFormat set the request format
JkRequestLogFormat "%w %V %T"
# Send all jsp requests to GlassFish
JkMount /*.jsp worker1
# Send all glassfish-test requests to GlassFish
```

```
JkMount /glassfish-test/* worker1
```

Example 6-5 `workers.properties` File for `Mod_jk`

This example shows a `workers.properties` that is set for `mod_jk`. This `workers.properties` file is referenced in the second line of example-6-4.

```
# Define 1 real worker using ajp13
worker.list=worker1
# Set properties for worker1 (ajp13)
worker.worker1.type=ajp13
worker.worker1.host=localhost
worker.worker1.port=8009
```

- For more information on Apache, see `http://httpd.apache.org/`.
- For more information on Apache Tomcat Connector, see `http://tomcat.apache.org/connectors-doc/index.html`.

To Load Balance Using `mod_jk` and Payara Server

Load balancing is the process of dividing the amount of work that a computer has to do between two or more computers so that more work gets done in the same amount of time. Load balancing can be configured with or without security.

In order to support stickiness, the Apache `mod_jk` load balancer relies on a `jvmRoute` system property that is included in any `JSESSIONID` received by the load balancer. This means that every Payara Server instance that is front-ended by the Apache load balancer must be configured with a unique `jvmRoute` system property.

1. On each of the instances, perform the steps in to-enable-mod_jk.
   If your instances run on the same machine, you must choose different JK ports. The ports must match `worker.worker*.port` in your `workers.properties` file. See the properties file in example-6-5.
2. On each of the instances, create the `jvmRoute` system property of Payara Server by using the "`create-jvm-options`" subcommand.
   Use the following format:

```
asadmin> create-jvm-options "-DjvmRoute=/instance-worker-name"/
```

where instance-worker-name is the name of the worker that you defined to represent the instance in the `workers.properties` file.

3. To apply your changes, restart Apache HTTP Server and Payara Server.

Example 6-6 `httpd.conf` File for Load Balancing

This example shows an `httpd.conf` file that is set for load balancing.

```
LoadModule jk_module /usr/lib/httpd/modules/mod_jk.so
JkWorkersFile /etc/httpd/conf/worker.properties
# Where to put jk logs
JkLogFile /var/log/httpd/mod_jk.log
# Set the jk log level [debug/error/info]
JkLogLevel debug
# Select the log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
# JkOptions indicate to send SSL KEY SIZE,
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories
# JkRequestLogFormat set the request format
JkRequestLogFormat "%w %V %T"
# Send all jsp requests to Payara Server
JkMount /*.jsp worker1
# Send all glassfish-test requests to Payara Server
JkMount /glassfish-test/* loadbalancer
```

Example 6-7 `workers.properties` File for Load Balancing

This example shows a `workers.properties` or `glassfish-jk.properties` file that is set for load balancing. The `worker.worker*.port` should match with JK ports you created.

```
worker.list=worker1,worker2,loadbalancer
worker.worker1.type=ajp13
worker.worker1.host=localhost
worker.worker1.port=8009
worker.worker1.lbfactor=1
worker.worker1.socket_keepalive=1
worker.worker1.socket_timeout=300
worker.worker2.type=ajp13
worker.worker2.host=localhost
worker.worker2.port=8010
worker.worker2.lbfactor=1
worker.worker2.socket_keepalive=1
worker.worker2.socket_timeout=300
worker.loadbalancer.type=lb
```

```
worker.loadbalancer.balance_workers=worker1,worker2
```

To Enable SSL between the `mod_jk` Load Balancer and the Browser

To activate security for `mod_jk` on Payara Server, you must first generate a Secure Socket Layer (SSL) self-signed certificate on the Apache HTTP Server with the `mod_ssl` module. The tasks include generating a private key, a Certificate Signing Request (CSR), a self-signed certificate, and configuring SSL-enabled virtual hosts.

Before You Begin, The `mod_jk` connector must be enabled.

1. Generate the private key as follows:

   ```
   openssl genrsa -des3 -rand file1:file2:file3:file4:file5 -out server.key 1024
   ```

   where `file1:file2:` and so on represents the random compressed files.

2. Remove the pass-phrase from the key as follows:

   ```
   openssl rsa -in server.key -out server.pem
   ```

3. Generate the CSR is as follows:

   ```
   openssl req -new -key server.pem -out server.csr
   ```

   Enter the information you are prompted for.

4. Generate a temporary certificate as follows:

```
openssl x509 -req -days 60 -in server.csr -signkey server.pem -out server.crt
```

This temporary certificate is good for 60 days. . Create the `http-ssl.conf` file under the `/etc/apache2/conf.d` directory. . In the `http-ssl.conf` file, add one of the following redirects: * Redirect a web application, for example, `JkMount /hello/* worker1`. * Redirect all requests, for example, `JkMount /* worker1`.

```
# Send all jsp requests to GlassFish
```

```
JkMount /*.jsp worker1
# Send all glassfish-test requests to GlassFish
JkMount /glassfish-test/* loadbalancer
```

Example 6-8 `http-ssl.conf` File for `mod_jk` Security

A basic SSL-enabled virtual host will appear in the `http-ssl.conf` file. In this example, all requests are redirected.

```
Listen 443
<VirtualHost _default_:443>
SSLEngine on
SSLCipherSuite ALL:!ADH:!EXP56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL
SSLCertificateFile "/etc/apache2/2.2/server.crt"
SSLCertificateKeyFile "/etc/apache2/2.2/server.pem"
JkMount /* worker1
</VirtualHost>
```

To Enable SSL between the `mod_jk` Load Balancer and Payara Server

This procedure does not enable SSL transfer between `mod_jk` and Payara Server. It enables `mod_jk` to forward SSL-encrypted information from the browser to Payara Server.

Before You Begin, The self-signed certificate must be configured.

1. Perform the steps in to-enable-mod_jk.
2. Start another Payara Server with at least one web application deployed. In order for the `mod_jk`-enabled network listener to start listening for requests, the web container must be started. Normally, this is achieved by deploying a web application.
3. Follow instructions from to-configure-an-http-listener-for-ssl on the `mod_jk` connector.
   Use the following format:

   ```
   asadmin> create-ssl --type http-listener --certname sampleCert new-listener
   ```

4. Add the following directives in the `httpd.conf` file under the `/etc/apache2/conf.d` directory:

   ```
   # Should mod_jk send SSL information (default is On)
   ```

```
JkExtractSSL On
# What is the indicator for SSL (default is HTTPS)
JkHTTPSIndicator HTTPS
# What is the indicator for SSL session (default is SSL_SESSION_ID)
JkSESSIONIndicator SSL_SESSION_ID
# What is the indicator for client SSL cipher suit (default is SSL_CIPHER )
JkCIPHERIndicator SSL_CIPHER
# What is the indicator for the client SSL certificated? (default is
SSL_CLIENT_CERT )
JkCERTSIndicator SSL_CLIENT_CERT
```

5. To apply your changes, restart Apache HTTP Server and Payara Server.

Administering `mod_proxy_ajp`

The Apache Connector `mod_proxy_ajp` can be used to connect the web container with Apache HTTP Server. By using `mod_proxy_ajp`, you can front Payara Server with Apache HTTP Server.

To Enable `mod_proxy_ajp`

You can front Payara Server with Apache HTTP Server and its `mod_proxy_ajp` connector by enabling the AJP protocol for one of Payara Server's network listeners, as described in this procedure. A typical use for `mod_proxy_ajp` would be to have Apache HTTP Server handle requests for static resources, while having requests for dynamic resources, such as servlets and JavaServer Pages (JSPs), forwarded to, and handled by the Payara Server back-end instance.

1. Install Apache HTTP Server. For information on installing Apache HTTP Server, see `https://httpd.apache.org/docs/2.4/install.html`.
2. Configure `apache2/conf/httpd.conf`, the main Apache configuration file. For example:

```
LoadModule proxy_module /usr/lib/httpd/modules/mod_proxy.so
LoadModule proxy_ajp_module /usr/lib/httpd/modules/mod_proxy_ajp.so

Listen 1979
NameVirtualHost *:1979
<VirtualHost *:1979>
   ServerName localhost
   ErrorLog /var/log/apache2/ajp.error.log
   CustomLog /var/log/apache2/ajp.log combined

   <Proxy *>
     AddDefaultCharset Off
```

```
      Order deny,allow
      Allow from all
    </Proxy>

    ProxyPass / ajp://localhost:8009/
    ProxyPassReverse / ajp://localhost:8009/
</VirtualHost>
```

3. Start Apache HTTP Server (`httpd`).
4. Create a jk-enabled network listener by using the `create-network-listener` subcommand.

```
asadmin> create-network-listener --protocol http-listener-1 \
--listenerport 8009 --jkenabled true jk-connector
```

5. If you expect to need more than five threads for the listener, increase the maximum threads in the `http-thread-pool` pool:

```
asadmin> set configs.config.server-config.thread-pools.thread-pool.\
http-thread-pool.max-thread-pool-size=value
```

6. To apply your changes, restart Payara Server.
   See to-restart-a-domain.
   - For more information on Apache, see `http://httpd.apache.org/`.
   - For more information on the Apache `mod_proxy_ajp` Connector, see https://httpd.apache.org/docs/2.4/mod/mod_proxy.html and https://httpd.apache.org/docs/2.4/mod/mod_proxy_ajp.html.
   - For more information on the AJP protocol, see http://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html.

To Load Balance Using `mod_proxy_ajp` and Payara Server

Load balancing is the process of dividing the amount of work that a computer has to do between two or more computers so that more work gets done in the same amount of time. In the Payara Server context, load balancing is most frequently used to distribute work among the instances in a Payara Server cluster.

To configure load balancing using `mod_proxy_ajp`, you must use the `mod_proxy_balancer` Apache module in addition to `mod_proxy_ajp`.

In order to support stickiness, the `mod_proxy_balancer` load balancer relies on a `jvmRoute` system property that is included in any `JSESSIONID` received by the load balancer. Consequently, every Payara Server instance that is front-ended by the Apache load balancer must be configured with a unique `jvmRoute` system property.

1. Install Apache HTTP Server. For information on installing Apache HTTP Server, see https://httpd.apache.org/docs/2.4/install.html
2. Configure `apache2/conf/httpd.conf`, the main Apache configuration file. For example:

```
LoadModule proxy_module /usr/lib/httpd/modules/mod_proxy.so
LoadModule proxy_ajp_module /usr/lib/httpd/modules/mod_proxy_ajp.so
LoadModule proxy_balancer_module /usr/lib/httpd/modules/mod_proxy_balancer.so

# Forward proxy needs to be turned off
ProxyRequests Off
# Keep the original Host Header
ProxyPreserveHost On

   <Proxy *>
      Order deny,allow
      Deny from all
      Allow from localhost
   </Proxy>

# Each BalancerMember corresponds to an instance in the Payara Server
# cluster. The port specified for each instance must match the ajp port
# specified for that instance.
<Proxy balancer://localhost>
    BalancerMember ajp://localhost:8009
    BalancerMember ajp://localhost:8010
    BalancerMember ajp://localhost:8011
</Proxy>
```

3. Start Apache HTTP Server (`httpd`).
4. In Payara Server, use the `create-network-listener` subcommand to create a jk-enabled network listener targeted to the cluster. For example:

```
asadmin> create-network-listener --jkenabled true --target cluster1 \
--protocol http-listener-1 --listenerport ${AJP_PORT} jk-listener
```

In this example, `cluster1` is the name of the cluster and `jk-listener` is the name of the new listener.

5. If you expect to need more than five threads for the listener, increase the maximum threads in the `http-thread-pool` pool:

```
asadmin> set configs.config.cluster1-config.thread-pools.thread-pool.\
http-thread-pool.max-thread-pool-size=value
```

6. Use the `create-jvm-options` subcommand to create the `jvmRoute` property targeted to the cluster. For example:

```
asadmin> create-jvm-options --target cluster1 \
"-DjvmRoute=\${AJP_INSTANCE_NAME}"
```

7. Use the `create-system-properties` subcommand to define the `AJP_PORT` and `AJP_INSTANCE_NAME` properties for each of the instances in the cluster, making sure to match the port values you used in Step 2 when specifying the load balancer members. For example:

```
asadmin> create-system-properties --target instance1 AJP_PORT=8009
asadmin> create-system-properties --target instance1 \
AJP_INSTANCE_NAME=instance1
asadmin> create-system-properties --target instance2 AJP_PORT=8010
asadmin> create-system-properties --target instance2 \
AJP_INSTANCE_NAME=instance2
asadmin> create-system-properties --target instance3 AJP_PORT=8011
asadmin> create-system-properties --target instance3 \
AJP_INSTANCE_NAME=instance3
```

   In this example, `instance1`, `instance2` and `instance3` are the names of the Payara Server instances in the cluster.

8. To apply your changes, restart Payara Server. See to-restart-a-domain.

Configuration Variables Reference

This section outlines how to reference different variable type sources in the configuration of a Payara Server domain.

References to System Properties

Variable references to system properties start with `${`, followed by the name of the variable, and end with `}`. An example of a variable reference that refers to a `java.home` system property is:

```
${java.home}
```

Values of system properties can be specified by:

- the JVM (standard JVM properties like `java.home`)
- Payara Server (a set of predefined variables, such as `com.sun.aas.instanceRoot`)
- the `create-system-properties` asadmin command (or in the Admin console `System properties` section)
- a JVM option prefixed with `-D`

When custom properties are specified, their value can also contain other variable references. It is therefore possible to define a system property using other system properties or environment variables.

References to Environment Variables

Environment variable references are used like system property references, but contain an extra `ENV` qualifier and equals sign before the variable name. An example of a reference to an environment variable `HOME` then looks like:

```
${ENV=HOME}
${ENV=MAYBE_AVAILABLE:default}
```

Environment variables need to be specified in the environment before a Payara instance is started. By adding a colon after the variable name, you may append a default value if the variable has not been set.

References to Password Aliases

References to password aliases are used to refer to a password which is stored

securely in the domain's keystore. They are treated in a special way and cannot be mixed with other variable references and with other characters in the same configuration value. They are only supposed to be used as a value anywhere a password is expected in the configuration.

Password aliases look like system property references, but contain an extra `ALIAS` qualifier and equals sign before the alias name. Whenever used, the configuration value can only contain a password alias and nothing more.

An example of a reference to a password alias called `cluster-password` for an attribute `admin-password` looks like:

```
admin-password="${ALIAS=cluster-password}"
```

Reference to Microprofile Config Properties

MicroProfile Configuration variable references can be used exactly like an environment variable reference, but using the `MPCONFIG` qualifier. An example of a reference to a MicroProfile Configuration variable called `foo` would look like:

```
${MPCONFIG=foo}
${MPCONFIG=bar:default}
```

MicroProfile Configuration variables must be declared in one of the Azul Payara config-sources.

NOTE By adding a colon (:) after the property name, you may append a default value if the variable has not been set.

Using Variable References

A variable can be referenced anywhere inside a string value of a property which supports variable references. Multiple references to system properties and environment variables can be used within the same configuration value, concatenating any number of variable references and plain text chunks. When a reference to a password alias is used, it has to expand to the whole property value and cannot be combined with any

other text or variable references.

Variable References in the Domain Configuration

Variable references can be used inside text values in the following configuration files within the domain directory:

- domain.xml (inside attribute values or inside text nodes that contain a configuration value)
- logging.properties (inside property values)

Variable References in Deployed Applications

Variable references can be used inside text values also in the following places within deployed applications and modules:

- Package descriptors:
  - web.xml / payara-web.xml / glassfish-web.xml (deprecated)
  - ejb-jar.xml / payara-ejb-jar.xml / glassfish-ejb-jar.xml (deprecated)
  - application.xml / payara-application.xml / glassfish-application.xml (deprecated)
  - payara-resources.xml / glassfish-resources.xml (deprecated)
- Framework descriptors:
  - persistence.xml
  - faces-config.xml
- In the annotations:
  - `@EJB`, `@WebService`, `@Stateless`, `@Stateful`, `@Singleton`, `@ActivationConfigProperty`, `@DataSource`, `@WebServlet`
- In JNDI lookups inside an application
- In pre and post boot command files.
- In JVM Options.
- In the `payara-expression-config.properties` file.

Examples

Here are several examples of where you can use variable references:

Within an Annotation

```
@DataSourceDefinition(name="java:global/ExampleDataSource",
      className="com.mysql.jdbc.jdbc2.optional.MysqlXADataSource",
      user="${mysql.user}",
      password="${ALIAS=mysql-db-password}",
      databaseName="${ENV=MYSQL_DBNAME}",
      serverName="localhost"
   )
```

Within a `web.xml` Descriptor

```
<data-source>
     <name>java:global/ExampleDataSource</name>
     <class-name>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</class-name>
     <server-name>localhost</server-name>
     <port-number>3306</port-number>
     <database-name>${ENV=MYSQL_DBNAME}</database-name>
     <user>${mysql.user}</user>
     <password>${ALIAS=mysql-db-password}</password>
</data-source>
```

Multiple Variable References Combined in a Single Value Withing the `domain.xml`

```
<domain log-root="${com.sun.aas.instanceRoot}/${ENV=STAGE_NAME}/logs">
```

Within a Pre or Post Boot Command File

```
set configs.config.server-config.http-service.virtual-server.server.default-web-
module=${ENV=deployed-app-name}
set configs.config.server-config.network-config.protocols.protocol.http-listener-
1.security-enabled=${ENV=security-enabled}
```

Within a `logging.properties` File

```
handlers=${ENV=CONSOLE-HANDLER}
handlerServices=${ENV=SYS-LOG-HANDLER}
```

Within a `payara-expression-config.properties` File

NOTE The `payara-expression-config.properties` config source is only available in Payara Server

```
fish.payara.example.system=${java.home}
```

```
fish.payara.example.env=${ENV=PATH}
fish.payara.example.password=${ALIAS=secret}
fish.payara.example.multiple=property containing both an ${ENV=envProperty} and a
${sysProp}
```

Subcommands for the `asadmin` Utility

For information and instructions on using the `asadmin` application deployment subcommands, see "Application Deployment^"

NOTE Online help for the `asadmin` subcommands can be invoked on the command line, for example, `asadmin create-domain --help`.

General Administration Subcommand

Subcommands	Description
`add-resources`	Creates the resources named in the specified XML file. Supported in remote mode only. For procedural information in this guide. see to-add-resources-from-an-xml-file
`asadmin`	Describes how the `asadmin` utility works.
`attach`	Attaches to subcommands that were started using the `asadmin --detach` option or that contain progress information. For procedural information in this guide, see to-run-asadmin-subcommands-in---detach-mode.
`configure-managed-jobs`	Configures how long information about subcommands (jobs) that were started using the `asadmin --detach` option or that contain progress information is kept. For procedural information in this guide, see to-run-asadmin-subcommands-in---detach-mode.
`create-module-config`	Adds the default configuration of a module to domain.xml.
`create-service`	Configures the starting of a domain administration server (DAS) on an unattended boot. On Oracle Solaris 10, this subcommand uses the Service Management Facility (SMF). For procedural information in this guide, see to-configure-a-das-or-an-instance-for-automatic-restart-on-oracle-solaris.
`create-system-properties`	Creates or updates system properties. Supported in remote mode only. For procedural information in this guide, see to-create-system-properties.
`delete-module-config`	Removes the configuration of a module from `domain.xml`.

Subcommands	Description
`delete-system-property`	Deletes system properties of a domain or configuration, or server instance. Supported in remote mode only. For procedural information in this guide, see to-delete-a-system-property.
`get`	Gets an attribute of an element in the `domain.xml` file. With the `-m` option, gets the names and values of the monitorable or configurable attributes. For procedural information in this guide, see guidelines-for-using-the-list-and-get-subcommands-for-monitoring.
`get-active-module-config`	Displays the current active configuration of a module.
`list`	Lists the configurable element. On Oracle Solaris, quotes are needed when running subcommands with `*` as the option value or operand. For procedural information in this guide, see guidelines-for-using-the-list-and-get-subcommands-for-monitoring.
`list-commands`	Lists all the `asadmin` subcommands, local subcommands first, then remote subcommands. You can specify that only remote subcommands or only local subcommands be displayed. Supported in remote mode only. For procedural information in this guide, see to-list-subcommands.
`list-containers`	Lists application containers and the status of each container. Supported in remote mode only. For procedural information in this guide, see to-list-containers.
`list-jobs`	Lists information about subcommands that were started using the `asadmin --detach` option or that contain progress information. For procedural information in this guide, see to-run-asadmin-subcommands-in---detach-mode.
`list-modules`	Lists modules that are accessible to the Payara Server subsystem. The status of each module is included. Supported in remote mode only. For procedural information in this guide, see to-list-modules.
`list-system-properties`	Lists the system properties of a domain or configuration. Supported in remote mode only. For procedural information in this guide, see to-list-system-properties.
`list-timers`	List the timers owned by a specific server instance. Supported in remote mode only. For procedural information in this guide, see to-list-timers.

Subcommands	Description
`multimode`	Provides an `asadmin>` prompt for running multiple subcommands while preserving options and environment settings. Supported in local mode only. For procedural information, see using-the-asadmin-utility.
`osgi`	Delegates the command line to the Apache Felix Gogo remote shell for the execution of OSGi shell commands. For procedural information in this guide, see to-run-apache-felix-gogo-remote-shell-commands.
`osgi-shell`	Provides interactive access to the Apache Felix Gogo remote shell for the execution of OSGi shell commands. For procedural information in this guide, see to-run-apache-felix-gogo-remote-shell-commands.
`set`	Sets the values of one or more configurable attributes. For procedural information in this guide, see configuring-monitoring.
`show-component-status`	Lists the status of existing components. Supported in remote mode only. For procedural information in this guide, see to-show-component-status.
`start-database`	Starts the internal H2 database server. Use this subcommand only for working with applications deployed to Payara Server. For procedural information in this guide, see to-start-the-database.
`stop-database`	Stops a process of the internal H2 database server. For procedural information in this guide, see to-stop-the-database.
`version`	Displays the version information for the option specified in archive or folder format. Supported in remote mode only. For procedural information in this guide, see to-display-the-payara-server-version.

Batch Jobs Subcommands

Subcommands	Description
`list-batch-jobs`	Lists batch jobs and job details. For procedural information in this guide, see to-list-batch-jobs.
`list-batch-job-executions`	Lists batch job executions and execution details. For procedural information in this guide, see to-list-batch-job-executions.
`list-batch-job-steps`	Lists steps for a specific batch job execution. For procedural information in this guide, see to-list-batch-job-steps.

Subcommands	Description
`list-batch-runtime-configuration`	Displays the configuration of the batch runtime. For procedural information in this guide, see to-list-the-batch-runtime-configuration.
`set-batch-runtime-configuration`	Configures the batch runtime. For procedural information in this guide, see to-configure-the-batch-runtime.

Concurrent Resources Subcommands

Subcommands	Description
`create-context-service`	Creates a context service resource. For procedural information in this guide, see to-create-a-context-service.
`create-managed-executor-service`	Creates a managed executor service resource. For procedural information in this guide, see to-create-a-managed-executor-service.
`create-managed-scheduled-executor-service`	Creates a managed scheduled executor service resource. For procedural information in this guide, see to-create-a-managed-scheduled-executor-service.
`create-managed-thread-factory`	Creates a managed thread factory resource. For procedural information in this guide, see to-create-a-managed-thread-factory.
`list-context-services`	Lists context service resources. For procedural information in this guide, see to-list-context-services.
`list-managed-executor-services`	Lists managed executor service resources. For procedural information in this guide, see to-list-managed-executor-services.
`list-managed-scheduled-executor-services`	Lists managed scheduled executor service resources. For procedural information in this guide, see to-list-managed-scheduled-executor-services.
`list-managed-thread-factories`	Lists managed thread factory resources. For procedural information in this guide, see to-list-managed-thread-factories.

Subcommands	Description
`delete-context-service`	Removes a context service resource. For procedural information in this guide, see to-delete-a-context-service.
`delete-managed-executor-service`	Removes a managed executor service resource. For procedural information in this guide, see to-delete-a-managed-executor-service.
`delete-managed-scheduled-executor-service`	Removes a managed scheduled executor service resource. For procedural information in this guide, see to-delete-a-managed-scheduled-executor-service.
`delete-managed-thread-factory`	Removes a managed thread factory resource. For procedural information in this guide, see to-delete-a-managed-thread-factory.

Connectivity Subcommands

Subcommands	Description
`create-admin-object`	Creates an administered object. For procedural information in this guide, see "To Create an Administered Object".
`create-connector-connection-pool`	Adds a new connector connection pool with the specified connection pool name. For procedural information in this guide, see "To Create a Connector Connection Pool".
`create-connector-resource`	Creates a connector resource. For procedural information in this guide, see "To Create a Connector Resource".
`create-connector-security-map`	Creates a connector security map for the specified connector connection pool. For procedural information, see "To Create a Connector Security Map".
`create-connector-work-security-map`	Creates a connector work security map for the specified resource adapter. Supported in remote mode only. For procedural information in this guide, see to-create-a-connector-work-security-map.
`create-jdbc-resource`	Creates a new JDBC resource. Supported in remote mode only. For procedural information in this guide, see to-create-a-jdbc-resource.

Subcommands	Description
`create-jdbc-connection-pool`	Registers a new JDBC connection pool with the specified JDBC connection pool name. Supported in remote mode only. For procedural information in this guide, see to-create-a-jdbc-connection-pool.
`create-resource-adapter-config`	Creates configuration information for the connector module. Supported in remote mode only. For procedural information in this guide, see "To Create Configuration Information for a Resource Adapter".
`delete-admin-object`	Deletes an administered object. For procedural information in this guide, see "To Delete an Administered Object".
`delete-connector-connection-pool`	Removes the connector connection pool specified using the `connector_connection_pool_name` operand. For procedural information in this guide, see "To Delete a Connector Connection Pool".
`delete-connector-resource`	Deletes connector resource. For procedural information in this guide, see "To Delete a Connector Resource".
`delete-connector-security-map`	Deletes a specified connector security map. Supported in remote mode only. For procedural information in this guide, see "To Delete a Connector Security Map".
`delete-connector-work-security-map`	Deletes a specified connector work security map. Supported in remote mode only. For procedural information in this guide, see "To Delete a Connector Work Security Map".
`delete-jdbc-connection-pool`	Deletes the specified JDBC connection pool. Supported in remote mode only. For procedural information in this guide, see to-delete-a-jdbc-connection-pool.
`delete-jdbc-resource`	Deletes a JDBC resource. The specified JNDI name identifies the resource to be deleted. Supported in remote mode only. For procedural information in this guide, see to-delete-a-jdbc-resource.
`delete-resource-adapter-config`	Deletes configuration information for the connector module. Supported in remote mode only. For procedural information in this guide, see to-delete-a-resource-adapter-configuration.
`flush-connection-pool`	Re-initializes all connections established in the specified connection. For procedural information in this guide, see to-reset-flush-a-connection-pool.

Subcommands	Description
`list-admin-objects`	Lists administered objects. For procedural information in this guide, see to-list-administered-objects.
`list-connector-connection-pools`	Lists the connector connection pools that have been created. For procedural information in this guide, see "To List Connector Connection Pools".
`list-connector-resources`	Creates connector resources. For procedural information in this guide, see "To List Connector Resources".
`list-connector-security-maps`	Lists the connector security maps belonging to a specified connector connection pool. For procedural information in this guide, see "To List Connector Security Maps".
`list-connector-work-security-maps`	Lists the existing connector work security maps for a resource adapter. Supported in remote mode only. For procedural information in this guide, see "o List Connector Work Security Maps".
`list-jdbc-connection-pools`	Lists the existing JDBC connection pools. Supported in remote mode only. For procedural information in this guide, see to-list-jdbc-connection-pools.
`list-jdbc-resources`	Lists the existing JDBC resources. Supported in remote mode only. For procedural information in this guide, see to-list-jdbc-resources.
`list-resource-adapter-configs`	Lists configuration information for the connector modules. Supported in remote mode only. For procedural information in this guide, see "To List Resource Adapter Configurations".
`ping-connection-pool`	Tests if a JDBC connection pool is usable. Supported in remote mode only. For procedural information in this guide, see to-contact-ping-a-connection-pool.
`update-connector-security-map`	Modifies a security map for the specified connector connection pool. For procedural information in this guide, see to-update-a-connector-security-map.
`update-connector-work-security-map`	Modifies a work security map that belongs to a specific resource adapter (connector module). For procedure information in this guide, see "To Update a Connector Work Security Map".

Domain Subcommands

Subcommands	Description
`backup-domain`	Describes how to back up a domain. Supported in local mode only. For procedural information in this guide, see to-back-up-a-domain.
`create-domain`	Creates the configuration of a domain. A domain can exist independent of other domains. Any user who has access to the `asadmin` utility on a given host can create a domain and store its configuration in a location of choice. For procedural information in this guide, see to-create-a-domain.
`delete-domain`	Deletes the specified domain. The domain must be stopped before it can be deleted. For procedural information in this guide, see to-delete-a-domain.
`list-backups`	Lists the existing domain backups. Supported in local mode only. For procedural information in this guide, see to-list-domain-backups.
`list-domains`	Lists the existing domains and their statuses. If the domain directory is not specified, the domains in the domain-root-dir, the default for which is as-install`/domains`, are displayed. For procedural information in this guide, see to-list-domains.
`login`	Allows you to log in to a domain. For procedural information in this guide, see to-log-in-to-a-domain.
`restart-domain`	Restarts the Domain Administration Server (DAS) of the specified domain. Supported in remote mode only. For procedural information in this guide, see to-restart-a-domain.
`restore-domain`	Recovers and domain from a backup file. Supported in local mode only. For procedural information in this guide, see to-restore-a-domain.
`start-domain`	Starts a domain. If the domain directory is not specified, the default `domain1` in the default domain-root-dir directory is started. If there are two or more domains, the domain_name operand must be specified. For procedural information in this guide, see to-start-a-domain.
`stop-domain`	Stops the domain administration server (DAS) of the specified domain. Supported in remote mode only. For procedural information in this guide, see to-stop-a-domain.

Subcommands	Description
`uptime`	Displays the length of time that the domain administration server (DAS) has been running since the last restart. Supported in remote mode only. For procedural information in this guide, see to-display-domain-uptime.

Internet Connectivity Subcommands

Subcommands	Description
`create-http`	Creates a set of HTTP parameters for a protocol, which in turn configures one or more network listeners. Supported in remote mode only. For procedural information in this guide, see create-an-http-configuration.
`create-http-listener`	Creates a new HTTP listener socket. Supported in remote mode only. For procedural information in this guide, see to-create-an-internet-connection.
`create-network-listener`	Creates a new HTTP listener socket. Supported in remote mode only. For procedural information in this guide, see to-create-an-internet-connection.
`create-protocol`	Creates a protocol for a listener. Supported in remote mode only. For procedural information in this guide, see create-a-protocol.
`create-transport`	Creates a transport object for a listener. Supported in remote mode only. For procedural information in this guide, see create-a-transport.
`create-virtual-server`	Creates the specified virtual server element. Supported in remote mode only. For procedural information in this guide, see create-a-virtual-server.
`create-ssl`	Creates and configures the SSL element in the selected HTTP listener to enable secure communication on that listener/service. Supported in remote mode only. For procedural information in this guide, see configure-an-http-listener-for-ssl.
`delete-http`	Deletes an existing HTTP configuration. Supported in remote mode only. For procedural information in this guide, see delete-an-http-configuration.
`delete-http-listener`	Deletes the specified HTTP listener. Supported in remote mode only. For procedural information in this guide, see to-delete-an-http-network-listener.
`delete-network-listener`	Deletes the specified HTTP listener. Supported in remote mode only. For procedural information in this guide, see to-delete-an-http-network-listener.

Subcommands	Description
`delete-protocol`	Deletes and existing HTTP protocol. Supported in remote mode only. For procedural information in this guide, see delete-a-protocol.
`delete-ssl`	Deletes the SSL element in the selected HTTP listener. Supported in remote mode only. For procedural information in this guide, see delete-ssl-from-an-http-listener.
`delete-transport`	Deletes and existing HTTP transport. Supported in remote mode only. For procedural information in this guide, see delete-a-transport.
`delete-virtual-server`	Deletes the specified virtual server element. Supported in remote mode only. For procedural information in this guide, see delete-a-virtual-server.
`list-http-listeners`	Lists the existing HTTP listeners. Supported in remote mode only. For procedural information in this guide, see to-list-http-network-listeners.
`list-network-listeners`	Lists the existing HTTP listeners. Supported in remote mode only. For procedural information in this guide, see to-list-http-network-listeners.
`list-protocols`	Lists the existing HTTP protocols. Supported in remote mode only. For procedural information in this guide, see list-protocols.
`list-transports`	Lists the existing HTTP transports. Supported in remote mode only. For procedural information in this guide, see list-transports.
`list-virtual-servers`	Lists the existing virtual servers. Supported in remote mode only. For procedural information in this guide, see to-list-virtual-servers.

Javamail Subcommands

Subcommands	Description
`create-javamail-resource`	Creates a JavaMail session resource. Supported in remote mode only. For procedural information in this guide, see to-create-a-javamail-resource.
`delete-javamail-resource`	Deletes a JavaMail session resource. Supported in remote mode only. For procedural information in this guide, see to-delete-a-javamail-resource.
`list-javamail-resources`	Creates JavaMail session resources. Supported in remote mode only. For procedural information in this guide, see to-list-javamail-resources.

JMS Subcommands

Subcommands	Description
`create-jmsdest`	Creates a JMS physical destination. Along with the physical destination, you use the `create-jms-resource` subcommand to create a JMS destination resource that has a `Name` property that specifies the physical destination. Supported in remote mode only. For procedural information in this guide, see to-create-a-jms-physical-destination.
`create-jms-host`	Creates a JMS host within the JMS service. Supported in remote mode only. For procedural information in this guide, see to-create-a-jms-host.
`create-jms-resource`	Creates a JMS connection factory resource or JMS destination resource. Supported in remote mode only. Supported in remote mode only. For procedural information in this guide, see to-create-a-connection-factory-or-destination-resource.
`delete-jmsdest`	Removes the specified JMS destination. Supported in remote mode only. For procedural information in this guide, see to-delete-a-jms-physical-destination.
`delete-jms-host`	Deletes a JMS host within the JMS service. Supported in remote mode only. For procedural information in this guide, see to-delete-a-jms-host.
`delete-jms-resource`	Deletes a JMS connection factory resource or JMS destination resource. Supported in remote mode only. For procedural information in this guide, see to-delete-a-connection-factory-or-destination-resource.
`flush-jmsdest`	Purges the messages from a physical destination in the specified JMS Service configuration of the specified target. Supported in remote mode only. For procedural information in this guide, see to-purge-messages-from-a-physical-destination.
`jms-ping`	Checks if the JMS service (also known as the JMS provider) is up and running. Supported in remote mode only. For procedural information in this guide, see troubleshooting-the-jms-service.
`list-jmsdest`	Lists the JMS physical destinations. Supported in remote mode only. For procedural information in this guide, see to-list-jms-physical-destinations.
`list-jms-hosts`	Lists the existing JMS hosts. Supported in remote mode only. For procedural information in this guide, see to-list-jms-hosts.

Subcommands	Description
`list-jms-resources`	Lists the existing JMS connection factory or destination resources. Supported in remote mode only. For procedural information in this guide, see to-list-jms-resources.

JNDI Subcommands

Subcommands	Description
`create-custom-resource`	Creates a custom JNDI resource. Supported in remote mode only. For procedural information in this guide, see to-create-a-custom-jndi-resource.
`create-jndi-resource`	Creates an external JNDI resource. Supported in remote mode only. For procedural information in this guide, see to-register-an-external-jndi-resource.
`delete-custom-resource`	Deletes a custom JNDI resource. Supported in remote mode only. For procedural information in this guide, see to-delete-a-custom-jndi-resource.
`delete-jndi-resource`	Deletes an external JNDI resource. Supported in remote mode only. For procedural information in this guide, see to-delete-an-external-jndi-resource.
`list-custom-resources`	Lists the existing custom JNDI resources. Supported in remote mode only. For procedural information in this guide, see to-list-custom-jndi-resources.
`list-jndi-entries`	Lists the entries in the JNDI tree. Supported in remote mode only. For procedural information in this guide, see to-list-external-jndi-entries.
`list-jndi-resources`	Lists the existing external JNDI resources. Supported in remote mode only. For procedural information in this guide, see to-list-external-jndi-resources.

JVM Subcommands

Subcommands	Description
`create-jvm-options`	Creates a JVM option in the Java configuration element of the `domain.xml` file. Supported in remote mode only. For procedural information in this guide, see to-create-jvm-options.
`delete-jvm-options`	Deletes the specified JVM option from the Java configuration element of the `domain.xml` file. Supported in remote mode only. For procedural information in this guide, see to-delete-jvm-options.

Subcommands	Description
`generate-jvm-report`	Generates a report showing the threads, classes, and memory for the virtual machine that runs Payara Server. For procedural information in this guide, see to-generate-a-jvm-report.
`list-jvm-options`	Lists the command-line options that are passed to the Java application launcher when Payara Server is started. Supported in remote mode only. For procedural information in this guide, see to-list-jvm-options.

Logging and Monitoring Subcommands

Subcommands	Description
`collect-log-files`	Collects all available log files and creates a ZIP archive. Supported in remote mode only. For procedural information in this guide, see to-collect-log-files-into-a-zip-archive.
`disable-monitoring`	Disables the monitoring service. Supported in remote mode only. For procedural information in this guide, see to-disable-monitoring.
`enable-monitoring`	Enables the monitoring service. Supported in remote mode only. For procedural information in this guide, see to-enable-monitoring.
`list-log-attributes`	Lists log file attributes. Supported in remote mode only. For procedural information in this guide, see configuring-the-logging-service.
`list-log-levels`	Lists the existing loggers. Supported in remote mode only. For procedural information in this guide, see to-list-log-levels.
`list-loggers`	Lists all public loggers in your distribution of Payara Server. Internal loggers are not listed. For procedural information in this guide, see to-list-loggers.
`monitor`	Displays monitoring information for the common Payara Server resources. Supported in remote mode only. For procedural information in this guide, see to-view-common-monitoring-data.
`rotate-log`	Rotates the `server.log` file and stores the old data in a time-stamped file. Supported in remote mode only. For procedural information in this guide, see to-rotate-log-files-manually.
`set-log-attributes`	Sets log file attributes. Supported in remote mode only. For procedural information in this guide, see configuring-the-logging-service.

Subcommands	Description
`set-log-file-format`	Sets the formatter used to format log records in log files. For procedural information in this guide, see setting-the-log-file-format.
`set-log-levels`	Sets the log level for a module. Supported in remote mode only. For procedural information in this guide, see setting-log-levels.

ORB Subcommands

Subcommands	Description
`create-iiop-listener`	Creates an IIOP listener. Supported in remote mode only. For procedural information in this guide, see to-create-an-iiop-listener.
`delete-iiop-listener`	Deletes an IIOP listener. Supported in remote mode only. For procedural information in this guide, see to-delete-an-iiop-listener.
`list-iiop-listeners`	Lists the existing IIOP listeners. Supported in remote mode only. For procedural information in this guide, see to-list-iiop-listeners.

Thread Pool Subcommands

Subcommands	Description
`create-threadpool`	Creates a new thread pool. Supported in remote mode only. For procedural information in this guide, see to-create-a-thread-pool.
`delete-threadpool`	Deletes the specified thread pool. Supported in remote mode only. For procedural information in this guide, see to-delete-a-thread-pool.
`list-threadpools`	Lists the existing thread pools. Supported in remote mode only. For procedural information in this guide, see to-list-thread-pools.

Transaction Service Subcommands

Subcommands	Description
`freeze-transaction-service`	Freezes the transaction subsystem during which time all the in-flight transactions are suspended. Supported in remote mode only. For procedural information, see to-stop-the-transaction-service.
`recover-transactions`	Manually recovers pending transactions. Supported in remote mode only. For procedural information, see to-manually-recover-transactions.

Subcommands	Description
`rollback-transaction`	Rolls back the named transaction. Supported in remote mode only. For procedural information, see to-roll-back-a-transaction.
`unfreeze-transaction-service`	Resumes all the suspended in-flight transactions. Invoke this subcommand on an already frozen transaction. Supported in remote mode only. For procedural information, see to-restart-the-transaction-service.

Mbeans Inventory

Below is a non-exhaustive list of MBeans which are useful to monitor in any Payara Server environment.

Other MBeans not listed show static information or are JDK-specific.

JVM Mbeans

Class Loading

In JMX Monitoring logging set the value to `java.lang:type=ClassLoading` to monitor this MBean

Attribute	Description	Comment
`loadedclass-count`	Number of classes currently loaded in the Java virtual machine	
`totalloadedclass-count`	Total number of classes that have been loaded since the Java virtual machine has started execution	
`unloadedclass-count`	Total number of classes unloaded since the Java virtual machine has started execution	Classes are only unloaded from Payara when the application they are in is un-deployed.

Compilation

In JMX Monitoring logging set the value to `java.lang:type=Compilation` to monitor this MBean

Attribute	Description	Comment
`totalcompilationtime-current`	Approximate accumulated elapsed time (in milliseconds) spent in compilation	

Garbage Collectors

In JMX Monitoring logging set the value to `java.lang:type=GarbageCollector,name=collectors-name` to monitor this MBean The garbage collector's name will be one of the following depending on your JDK configuration options:

- Copy
- G1 Young Generation
- G1 Mixed Generation
- MarkSweepCompact
- ParNew
- PS MarkSweep
- PS Scavenge

Attribute	Description	Comment
`collectioncount-count`	Total number of collections that have occurred	The number of times this garbage collector has been called since Payara Server was started
`collectiontime-count`	Approximate accumulated collection elapsed time in milliseconds	How much time has been spent on garbage collection since Payara Server was started

Memory

In JMX Monitoring logging set the value to `java.lang:type=Memory` to monitor this MBean

Attribute	Description	Comment
`ObjectsPendingFinalization`	Approximate number of objects for which finalization is pending	
`UsedHeapSize`	Amount of used memory in bytes	
`UsedNonHeapSize`	Amount of used memory in bytes	

Runtime

In JMX Monitoring logging set the value to `java.lang:type=Runtime` to monitor this MBean.

Attribute	Description	Comment
`uptime-count`	Uptime of the Java virtual machine in milliseconds	

Thread System

In JMX Monitoring logging set the value to `java.lang:type=Threading` to monitor this MBean.

Attribute	Description	Comment
`currentthreadcputime`	Returns the total CPU time for the current thread in nanoseconds	
`currentthreadusertime`	Returns the CPU time that the current thread has executed in user mode in nanoseconds	
`daemonthreadcount`	Returns the current number of live daemon threads	
`deadlockedthreads`	Finds cycles of threads that are in deadlock waiting to acquire object monitors	Is either a list of deadlocked threads or the sentence "None of the threads are monitor deadlocked."

Attribute	Description	Comment
`monitordeadlockedthreads`	Finds cycles of threads that are in deadlock waiting to acquire object monitors	Is either a list of deadlocked threads or the sentence "None of the threads are monitor deadlocked."
`peakthreadcount`	Returns the peak live thread count since the Java virtual machine started or peak was reset	
`threadcount`	Returns the current number of live threads including both daemon and non-daemon threads	
`totalstartedthreadcount`	Returns the total number of threads created and also started since the Java virtual machine started	

Azul Payara Mbeans

These MBeans are registered exclusively by Payara Server and provide additional information about current usage.

They are all in the form of `amx:pp=amx:pp=/mon/server-mon[instance-name],type=type-of-mbean,name=MBeanName` where `instance-name` is replaced by the name of the instance i.e. `amx:pp=/mon/server-mon[server],type=web-request-mon,name=web/request`.

See enable-jmx for how to view all available MBeans.

IMPORTANT	Azul Payara MBeans are only available when AMX is booted. This is done automatically once a JMX client connects. For all other cases, AMX must be either configured to boot at server startup (through domain config) or booted via a JMX operation `bootAMX` on the `amx-support:type=boot-amx` MBean using a non-JMX

> client (e.g. over a local socket).
> Programmatic activation during application deployment (by means of a Startup EJB singleton for example) will not work.

Connection Queue

Network Connection Queue Statistics

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=connection-queue-mon,name=network//connection-queue` to use this MBean.

To monitor a specific network connection set the name value to be `network/name-of-listener/connection-queue` i.e. `name=network/http-listener-1/connection-queue`.

Attribute	Description	Comment
`peakqueued`	Largest number of connections that were in the queue simultaneously	
`countoverflows`	Number of times the queue has been too full to accommodate a connection	
`counttotalconnections`	Total number of connections that have been accepted	
`countopenconnections`	The number of open/active connections	
`countqueued`	Number of connections currently in the queue	
`countqueued1minuteaverage`	Average number of connections queued in the last 1 minute	
`countqueued5minutesaverage`	Average number of connections queued in the last 5 minutes	

Attribute	Description	Comment
`countqueued15minutesaverage`	Average number of connections queued in the last 15 minutes	
`counttotalqueued`	Total number of connections that have been queued	

Connector Service

Connector Container Work Management Statistics

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=connector-service-mon,name=jms-service/work-management` to monitor this MBean.

Attribute	Description	Comment
`activeworkcount`	Number of active work objects	
`workrequestwaittime`	Wait time of a work object before it gets executed	
`waitqueuelength`	Number of work objects waiting in the queue for execution	
`rejectedworkcount`	Number of work objects rejected by the application server	
`submittedworkcount`	Number of work objects submitted by a connector module for execution	
`completedworkcount`	Number of work objects completed execution	

Deployment

Deployment Module Statistics

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=deployment-mon,name=deployment/lifecycle` to monitor this MBean.

Attribute	Description	Comment
`activeapplicationsdeployedcount`	Number of applications deployed	
`totalapplicationsdeployedcount`	Total number of applications ever deployed	This does not persist across restarts

EJB Security

EJB Security Deployment Statistics

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=ejb-security-mon,name=security/ejb` to monitor this MBean.

Attribute	Description	Comment
`policyconfigurationcount`	Count of EJB policy configurations	
`securitymanagercount`	Count of EJB security managers	

Thread Pool Executor

`ThreadPoolExecutor` Statistics

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=exec-pool-mon,name=ejb/default-exec-pool` to monitor this MBean.

Attribute	Description	Comment
`corenumthreads`	Core number of threads in the associated pool	
`maxnumthreads`	Maximum number of threads in the associated pool	
`numthreads`	Current number of threads in the associated pool	
`activenumthreads`	Number of active threads in the associated pool	
`totaltaskscreated`	Number of tasks created in the associated pool	

Attribute	Description	Comment
`keepalivetime`	Keep-Alive time for threads in the associated pool	
`numtaskscompleted`	Number of tasks completed in the associated pool	
`largestnumthreads`	Largest number of simultaneous threads in the associated pool	

JDBC Connection Pool

JDBC Connection Statistics

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=jdbc-connection-pool-mon,name=resources/NameOfPool` to monitor this MBean, replacing `NameOfPool` to whatever the actual name of the pool is i.e. **H2Pool**.

Attribute	Description	Comment
`numconncreated`	The number of physical connections that were created since the last reset	
`numconndestroyed`	Number of physical connections that were destroyed since the last reset.	
`numconnfree`	The total number of free connections in the pool as of the last sampling	
`numpotentialconnleak`	Number of potential connection leaks	
`numconnfailedvalidation`	The total number of connections in the connection pool that failed validation from the start time until the last sample time	

Attribute	Description	Comment
`connrequestwaittime`	The longest and shortest wait times of connection requests. The current value indicates the wait time of the last request that was serviced by the pool.	Unit is milliseconds
`numconnacquired`	Number of logical connections acquired from the pool	
`numconnreleased`	Number of logical connections released to the pool	
`averageconnwaittime`	Average wait-time-duration per successful connection request	Unit is milliseconds
`numconnsuccessfullymatched`	Number of connections successfully matched	
`numconnnotsuccessfullymatched`	Number of connections rejected during matching	
`waitqueuelength`	Number of connection requests in the queue waiting to be serviced	
`numconntimedout`	The total number of connections in the pool that timed out between the start time and the last sample time	

Keep Alive

Keep-Alive Statistics

This MBean is for network connections in keep-alive mode. For more details on `keep-alive` see RFC 7230 6.3.

NOTE	As of **HTTP 1.1** all connections are `keep-alive` unless declared otherwise.

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=keep-alive-mon,name=network//keep-alive` to use this MBean. To monitor a specific network connection set the name value to be `network/name-of-listener/keep-alive` i.e. `name=network/http-listener-1/keep-alive`.

Attribute	Description	Comment
`counttimeouts`	Number of keep-alive connections that timed out	
`counthits`	Number of requests received by connections in keep-alive mode	
`countconnections`	Number of connections in keep-alive mode	
`countflushes`	Number of keep-alive connections that were closed	
`countrefusals`	Number of keep-alive connections that were rejected	

Managed Executor Service

`ManagedExecutorService` Statistics

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=managed-executor-service-mon,name=executorService/concurrent/NameOfManagedExecutorService` to use this MBean, replacing `NameOfManagedExecutorService` to whatever the actual name of the service is i.e. `__defaultManagedExecutorService`.

Attribute	Description	Comment
`PoolSize`	The current number of threads in the pool	
`ActiveCount`	The approximate number of active threads	
`CompletedTaskCount`	Number of tasks completed	

Attribute	Description	Comment
`LargestPoolSize`	The largest number of threads that have ever simultaneously been in the pool	
`TaskCount`	The number of tasks executed by the executor service	

Message Driven Beans

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=message-driven-bean-mon,name=applicationName/ClassUsingBean` to use this MBean, replacing `applicationName` with the name of your application using JMS and `ClassUsingBean` with the class that has the `@MessageDriven` annotation on it.

Attribute	Description	Comment
`createcount`	Number of times EJB create method is called	
`removecount`	Number of times EJB remove method is called	
`messagecount`	Number of messages received for the message-driven bean	

Request

Web Container HTTP Service Statistics

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=request-mon,name=http-service/VirtualServer/request` to use this MBean, replacing `VirtualServer` name of the virtual server it is running on.

This MBean differs from Connection Queue statistics by being selected by virtual server rather than by listener, as well as providing some additional information about responses.

Attribute	Description	Comment
`method`	The method of the last request serviced	This will be one of `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `TRACE`, `OPTIONS`, `CONNECT` or `PATCH`
`countopenconnections`	The number of open connections	Unlike most other attributes with count in the name, this one is the current number rather than total over the server lifetime
`countrequests`	The number of requests received	The number of requests received since the server was started
`uri`	The URI of the last request serviced	
`maxtime`	Longest response time for a request; not a cumulative value, but the largest response time from among the response times	
`count200`	Number of responses with a status code equal to **200**	This is the total since the server started
`count2xx`	Number of responses with a status code in the **2xx** range	This is the total since the server started
`count302`	Number of responses with a status code equal to **302**	This is the total since the server started
`count304`	Number of responses with a status code equal to **304**	This is the total since the server started
`count3xx`	Number of responses with a status code in the **3xx** range	This is the total since the server started
`count400`	Number of responses with a status code equal to **400**	This is the total since the server started
`count401`	Number of responses with a status code equal to **401**	This is the total since the server started

Attribute	Description	Comment
`count403`	Number of responses with a status code equal to **403**	This is the total since the server started
`count404`	Number of responses with a status code equal to **404**	This is the total since the server started
`count4xx`	Number of responses with a status code in the **4xx** range	This is the total since the server started
`count503`	Number of responses with a status code in the **5xx** range	This is the total since the server started
`countother`	Number of responses with a status code outside the **2xx**, **3xx**, **4xx**, and **5xx** range	This is the total since the server started
`countbytestransmitted`	The number of bytes transmitted	
`countbytesreceived`	The number of bytes received	
`errorcount`	Cumulative value of the error count, with error count representing the number of cases where the response code was greater than or equal to **400**	
`processingtime`	Average request processing time	Unit is milliseconds

Security Realm

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=security-realm-mon,name=security/realm` to use this MBean.

Attribute	Description	Comment
`RealmCount`	Security Realm Count	

Server

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=server-`

`runtime-mon` to use this MBean.

Attribute	Description	Comment
`state`	state of the server such as **Running**, **Stopped**, **Failed**	
`uptime`	uptime of the Java virtual machine in milliseconds	

Servlet

Web Container Servlet Statistics

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=servlet-mon,name=ApplicationName/InstanceName/ServletName` to use this MBean, where `ApplicationName` is the name of your application and `InstanceName` is the instance it is running.

This MBean provides information on all servlets within an application.

Attribute	Description	Comment
`totalservletsloadedcount`	Total number of Servlets ever loaded	
`activeservletsloadedcount`	Number of Servlets loaded	
`servletprocessingtimes`	Cumulative Servlet processing times	Unit is milliseconds

Web Container Servlet Instance Statistics

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=servlet-instance-mon,name=ApplicationName/VirtualServerName/ServletName` to use this MBean, where `ApplicationName` is the name of your application, `VirtualServerName` is the virtual server it is running on and `ServletName` is the name of the servlet to access.

If the servlet is annotation with `@WebServlet(name = "ServletName")` then the

servlet name will be the name in the annotation, otherwise it is the fully qualified class name.

Attribute	Description	Comment
`errorcount`	Number of error responses (that is, responses with a status code greater than or equal to 400)	
`requestcount`	Number of requests processed	
`processingtime`	Average response time	Unit is milliseconds
`maxtime`	Maximum response time	Unit is milliseconds
`servicetime`	Aggregate response time	

Singleton Bean

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=singleton-bean-mon,name=ApplicationsName/ClassName` where `ApplicationName` is the name of your application and `ClassName` of the name of the Singleton EJB class.

Attribute	Description	Comment
`createcount`	Number of times EJB create method is called	
`removecount`	Number of times EJB remove method is called	

Stateful Bean

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=stateful-bean-mon,name=ApplicationsName/ClassName` where `ApplicationName` is the name of your application and `ClassName` of the name of the Stateful EJB class.

Attribute	Description	Comment
`createcount`	Number of times EJB create method is called	
`removecount`	Number of times EJB remove method is called	
`methodreadycount`	Number of stateful session beans in `MethodReady` state	
`passivecount`	Number of stateful session beans in Passive state	

Stateless Bean

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=stateless-bean-mon,name=ApplicationsName/ClassName` where `ApplicationName` is the name of your application and `ClassName` of the name of the Stateless EJB class.

Attribute	Description	Comment
`createcount`	Number of times EJB create method is called	
`removecount`	Number of times EJB remove method is called	
`methodreadycount`	Number of stateful session beans in `MethodReady` state	

Thread Pool

Thread Pool Statistics

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=thread-pool-mon,name=network/NetworkListenerName/thread-pool` where `NetworkListenerName` is the name of the network listener to monitor.

Alternatively set the value to `amx:pp=/mon/server-mon[server],type=thread-pool-mon,name=network//global-thread-pool-stats` for totals across all

thread pools.

Attribute	Description	Comment
`corethreads`	Core number of threads in the thread pool	
`totalexecutedtasks`	Provides the total number of tasks, which were executed by the thread pool	
`maxthreads`	Maximum number of threads allowed in the thread pool	
`currentthreadcount`	Provides the number of request processing threads currently in the listener thread pool	
`currentthreadsbusy`	Provides the number of request processing threads currently in use in the listener thread pool serving requests	

Transaction Service

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=transaction-service-mon,name=transaction-service` to use this MBean.

Attribute	Description	Comment
`activecount`	Provides the number of transactions that are currently active.	
`committedcount`	Provides the number of transactions that have been committed.	
`rolledbackcount`	Provides the number of transactions that have been rolled back.	

Attribute	Description	Comment
`state`	Indicates if the transaction service has been frozen.	Returns `false` if service is working fine

Virtual Server

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=virtualserverinfo-mon,name=http-service/VirtualServerName` where `VirtualServerName` is the name of the virtual server.

Attribute	Description	Comment
`id`	The id of the virtual server	
`hosts`	The host (alias) names of the virtual server"	
`mode`	The mode of the virtual server	Is either active or unknown
`state`	The state of the virtual server	

Web Request

In JMX Logging set value to `amx:pp=/mon/server-mon[server],type=web-request-mon,name=ApplicationName/VirtualServer` where `ApplicationName` is the name of your application and `VirtualServer` is the name of the virtual server it is running on. There is also the special value of `web/request` for all requests to every virtual server and application.

Attribute	Description	Comment
`errorcount`	Cumulative value of the error count, with error count representing the number of cases where the response code was greater than or equal to **400**	
`requestcount`	Cumulative number of requests processed so far	

Attribute	Description	Comment
`processingtime`	Average request processing time	
`maxtime`	Longest response time for a request; not a cumulative value, but the largest response time from among the response times	

Openmq Mbeans

Payara Server includes OpenMQ as a JMS broker, which includes its own MBeans.Documentation on them can be found in the official Eclipse Open Message Queue Developer's Guide for Java Clients.

Deployment Planning

Overview of Payara Server Deployment Planning

This section explains how to build a production deployment of Payara Server.

Product Concepts

Payara Server provides a robust platform for the development, deployment, and management of Jakarta EE applications. Key features include scalable transaction management, web services performance, clustering, security, and integration capabilities.

Jakarta EE Platform Overview

Payara Server implements Jakarta EE technology. The Jakarta EE platform is a set of standard specifications that describe application components, APIs, and the runtime containers and services of an application server.

Jakarta EE Applications

Jakarta EE applications are made up of components such as JavaServer Pages (JSP), Java servlets, and Enterprise JavaBeans (EJB) modules. These components enable software developers to build large-scale, distributed applications. Developers package Jakarta EE applications in Java Archive (JAR) files (similar to zip files), which can be distributed to production sites. Administrators install Jakarta EE applications onto

Payara Server by deploying Jakarta EE JAR files onto one or more server instances (or clusters of instances).

Containers

Each server instance includes two containers: web and EJB. A container is a runtime environment that provides services such as security and transaction management to Jakarta EE components. Web components, such as Java Server Pages and servlets, run within the web container. Enterprise JavaBeans run within the EJB container.

Jakarta EE Services

The Jakarta EE platform provides services for applications, including:

- Naming - A naming and directory service binds objects to names. A Jakarta EE application can locate an object by looking up its Java Naming and Directory Interface (JNDI) name.
- Security - The Java Authorization Contract for Containers (JACC) is a set of security contracts defined for the Jakarta EE containers. Based on the client's identity, containers can restrict access to the container's resources and services.
- Transaction management - A transaction is an indivisible unit of work. For example, transferring funds between bank accounts is a transaction. A transaction management service ensures that a transaction is either completed, or is rolled back.
- Message Service - Applications hosted on separate systems can communicate with each other by exchanging messages using the Java Message Service (JMS). JMS is an integral part of the Jakarta EE platform and simplifies the task of integrating heterogeneous enterprise applications.

Web Services

Clients can access a Jakarta EE application as a remote web service in addition to accessing it through HTTP, RMI/IIOP, and JMS. Web services are implemented using the Java API for XML-based web services (JAX-WS). A Jakarta EE application can also act as a client to web services, which would be typical in network applications.

Web Services Description Language (WSDL) is an XML format that describes web service interfaces. Web service consumers can dynamically parse a WSDL document to determine the operations a web service provides and how to execute them. Payara Server distributes web services interface descriptions using a registry that other applications can access through the Java API for XML Registries (JAX-R).

Client Access

Clients can access Jakarta EE applications in several ways. Browser clients access web applications using hypertext transfer protocol (HTTP). For secure communication, browsers use the HTTP secure (HTTPS) protocol that uses secure sockets layer (SSL).

Rich client applications running in the Application Client Container can directly look up and access Enterprise JavaBeans using an Object Request Broker (ORB), Remote Method Invocation (RMI) and the internet inter-ORB protocol (IIOP), or IIOP/SSL (secure IIOP). They can access applications and web services using HTTP/HTTPS, JMS, and JAX-WS. They can use JMS to send messages to and receive messages from applications and message-driven beans.

Clients that conform to the Web Services-Interoperability (WS-I) Basic Profile can access Jakarta EE web services. WS-I is an integral part of the Jakarta EE standard and defines interoperable web services. It enables clients written in any supporting language to access web services deployed to Payara Server.

The best access mechanism depends on the specific application and the anticipated volume of traffic. Payara Server supports separately configurable listeners for HTTP, HTTPS, JMS, IIOP, and IIOP/SSL. You can set up multiple listeners for each protocol for increased scalability and reliability.

Jakarta EE applications can also act as clients of Jakarta EE components such as Enterprise JavaBeans modules deployed on other servers, and can use any of these access mechanisms.

External Systems and Resources

On the Jakarta EE platform, an external system is called a resource. For example, a

database management system is a JDBC resource. Each resource is uniquely identified and by its Java Naming and Directory Interface (JNDI) name. Applications access external systems through the following APIs and components:

- Java Database Connectivity (JDBC) - A database management system (DBMS) provides facilities for storing, organizing, and retrieving data. Most business applications store data in relational databases, which applications access via JDBC. Payara Server includes the Java DB database for use sample applications and application development and prototyping, though it is not suitable for deployment. Payara Server provides certified JDBC drivers for connecting to major relational databases. These drivers are suitable for deployment.
- Java Message Service - Messaging is a method of communication between software components or applications. A messaging client sends messages to, and receives messages from, any other client via a messaging provider that implements the Java Messaging Service (JMS) API. Payara Server includes a high-performance JMS broker, Open Message Queue.
- Jakarta EE Connectors - The Jakarta EE Connector architecture enables integrating Jakarta EE applications and existing Enterprise Information Systems (EIS). An application accesses an EIS through a portable Jakarta EE component called a connector or resource adapter, analogous to using JDBC driver to access an RDBMS. Resource adapters are distributed as standalone Resource Adapter Archive (RAR) modules or included in Jakarta EE application archives. As RARs, they are deployed like other Jakarta EE components. Payara Server includes evaluation resource adapters that integrate with popular EIS.
- JavaMail - Through the JavaMail API, applications can connect to a Simple Mail Transport Protocol (SMTP) server to send email and to an IMAP or POP3 server to receive email.

Payara Server Components

This section describes the components in Payara Server.

The administration tools, such as the browser-based Administration Console,

communicate with the domain administration server (DAS), which in turn communicates with the server instances.

Server Instances

A server instance is a Payara Server running in a single Java Virtual Machine (JVM) process. The supported platforms for Payara Server are described in this following article: "Supported Platforms".

It is usually sufficient to create a single server instance on a machine, since Payara Server and accompanying JVM are both designed to scale to multiple processors. However, it can be beneficial to create multiple instances on one machine for application isolation and rolling upgrades. In some cases, a large server with multiple instances can be used in more than one administrative domain. The administration tools makes it easy to create, delete, and manage server instances across multiple machines.

Administrative Domains

An administrative domain (or simply domain) is a group of server instances that are administered together. A server instance belongs to a single administrative domain. The instances in a domain can run on different physical hosts.

You can create multiple domains from one installation of Payara Server. By grouping server instances into domains, different organizations and administrators can share a single Payara Server installation. Each domain has its own configuration, log files, and application deployment areas that are independent of other domains. Changing the configuration of one domain does not affect the configurations of other domains. Likewise, deploying an application on one domain does not deploy it or make it visible to any other domain.

NOTE	All hosts in a domain on which the DAS and Payara Server instances are running must have the same operating system.

Domain Administration Server (DAS)

A domain has one Domain Administration Server (DAS), a specially designated Payara Server instance that hosts the administrative applications. The DAS authenticates the administrator, accepts requests from administration tools, and communicates with server instances in the domain to carry out the requests.

The administration tools are the `asadmin` command-line tool and the browser-based Administration Console. Payara Server also provides a REST-ful API for server administration. The administrator can view and manage a single domain at a time, thus enforcing secure separation.

The DAS is also sometimes referred to as the admin server or default server. It is referred to as the default server because it is the default target for some administrative operations.

Since the DAS is a Payara Server instance, it can also host Jakarta EE applications for testing purposes. However, do not use it to host production applications. You might want to deploy applications to the DAS, for example, if the clusters and instances that will host the production application have not yet been created.

The DAS keeps a repository containing the configuration of its domain and all the deployed applications. If the DAS is inactive or down, there is no impact on the performance or availability of active server instances, however administrative changes cannot be made. In certain cases, for security purposes, it may be useful to intentionally stop the DAS process, for example to reboot the host operating system to install a kernel patch or a hardware upgrade.

Administrative commands are provided in order to back up and restore the domain configuration and applications. With the standard backup and restore procedures, you can quickly restore working configurations. If the DAS host fails, you must create a new DAS installation to restore the previous domain configuration. For instructions, see "administering-domains" in the Payara Server General Administration section.

Clusters

A cluster is a named collection of server instances that share the same applications, resources, and configuration information. You can group server instances on different machines into one logical cluster and administer them as one unit. You can easily control the lifecycle of a multi-machine cluster with the DAS.

Clusters enable horizontal scalability, load balancing, and failover protection. By definition, all the instances in a cluster have the same resource and application configuration. When a server instance or a machine in a cluster fails, the load balancer detects the failure, redirects traffic from the failed instance to other instances in the cluster, and recovers the user session state. Since the same applications and resources are on all instances in the cluster, an instance can fail over to any other instance in the cluster.

NOTE	All hosts in a cluster on which the DAS and Payara Server instances are running must have the same operating system.

Clusters, domains, and instances are related as follows:

- An administrative domain can have zero or more clusters.
- A cluster has one or more server instances.
- A cluster belongs to a single domain.

Named Configurations

A named configuration is an abstraction that encapsulates Payara Server property settings. Clusters and stand-alone server instances reference a named configuration to get their property settings. With named configurations, Jakarta EE container configurations are independent of the physical machine on which they reside, except for particulars such as IP address, port number, and amount of heap memory. Using named configurations provides power and flexibility to Payara Server administration.

To apply configuration changes, you simply change the property settings of the named configuration, and all the clusters and stand-alone instances that reference it pick up the

changes. You can only delete a named configuration when all references to it have been removed. A domain can contain multiple named configurations.

Payara Server comes with a default configuration, called `default-config`. The default configuration is optimized for developer productivity.

You can create your own named configuration based on the default configuration that you can customize for your own purposes. Use the Administration Console and `asadmin` command line utility to create and manage named configurations.

Session Persistence

Jakarta EE applications typically have significant amounts of session state data. A web shopping cart is the classic example of a session state. Also, an application can cache frequently-needed data in the session object. In fact, almost all applications with significant user interactions need to maintain a session state. Both HTTP sessions and stateful session beans (SFSBs) have session state data.

While the session state is not as important as the transactional state stored in a database, preserving the session state across server failures can be important to end users.

Payara Server provides the capability to save, or persist, this session state in a repository. If the Payara Server instance that is hosting the user session experiences a failure, the session state can be recovered. The session can continue without loss of information.

Payara Server supports the following session persistence types:

- `memory`
- `hazelcast` (Data Grid)
- `file`

With `memory` persistence, the state is always kept in memory and does not survive failure.

With `hazelcast` persistence, Payara Server stores session data in the Data Grid, implemented internally using a Hazelcast cluster. For SFSBs, if `hazelcast` persistence is not specified, Payara Server stores state information in the `session-store` subdirectory of its corresponding instance.

Checking an SFSB state for changes that need to be saved is called **checkpointing**. When enabled, checkpointing generally occurs after any transaction involving the SFSB is completed, even if the transaction rolls back. For more information on developing stateful session beans,see "using-session-beans" in the Payara Server Application Development section. For more information on enabling SFSB failover, see "stateful-session-bean-failover" in the Payara Server High Availability section.

Apart from the number of requests being served by Payara Server, the session persistence configuration settings also affect the session information in each request.

For more information on configuring session persistence, see "configuring-high-availability-session-persistence-and-failover" in the Payara Server High Availability section.

IIOP Load Balancing in a Cluster

With IIOP load balancing, IIOP client requests are distributed to different server instances or name servers. The goal is to spread the load evenly across the cluster, thus providing scalability. IIOP load balancing combined with EJB clustering and availability features in Payara Server provides not only load balancing but also EJB failover.

There are two steps to IIOP failover and load balancing. The first step, bootstrapping, is the process by which the client sets up the initial naming context with one ORB in the cluster. The client attempts to connect to one of the IIOP endpoints. When launching an application client using the `appclient` script, you specify these endpoints using the `-targetserver` option on the command line or `target-server` elements in the `sun-acc.xml` deployment descriptor. The client randomly chooses one of these endpoints and tries to connect to it, trying other endpoints if needed until one works.

The second step concerns sending messages to a specific EJB. By default, all naming

look-ups, and therefore all EJB accesses, use the cluster instance chosen during bootstrapping. The client exchanges messages with an EJB through the client ORB and server ORB. As this happens, the server ORB updates the client ORB as servers enter and leave the cluster. Later, if the client loses its connection to the server from the previous step, the client fails over to some other server using its list of currently active members. In particular, this cluster member might have joined the cluster after the client made the initial connection.

When a client performs a JNDI lookup for an object, the Naming Service creates an InitialContext` (IC) object associated with a particular server instance. From then on, all lookup requests made using that IC object are sent to the same server instance. All `EJBHome` objects looked up with that `InitialContext` are hosted on the same target server. Any bean references obtained henceforth are also created on the same target host. This effectively provides load balancing, since all clients randomize the list of live target servers when creating `InitialContext` objects. If the target server instance goes down, the lookup or EJB method invocation will fail over to another server instance.

Adding or deleting new instances to the cluster does not update the existing client's view of the cluster. You must manually update the endpoints list on the client side.

Message Queue and JMS Resources

The Open Message Queue (MQ) software provides reliable, asynchronous messaging for distributed applications. Message Queue is an enterprise messaging system that implements the Java Message Service (JMS) standard. Message Queue provides messaging for Jakarta EE application components such as message-driven beans (MDBs).

Payara Server implements the Java Message Service (JMS) API by integrating Message Queue into Payara Server. Payara Server includes the Enterprise version of Message Queue which has failover, clustering and load balancing features.

For basic JMS administration tasks, use the Administration Console and `asadmin`

command-line utility.

For advanced tasks, including administering a Message Queue cluster, use the tools provided in the `as-install/mq/bin` directory.

For information on deploying JMS applications and Message Queue clustering for message failover, see planning-your-deployment.

Configuration Roadmap for High Availability of Payara Server

The following procedure lists the major tasks for configuring Payara Server for high availability. The procedure also provides cross-references to detailed instructions for performing each task.

To Configure Payara Server for High Availability

1. Determine your requirements and goals for performance and QoS. For more information, see the following documentation:
    - establishing-performance-goals
    - planning-the-network-configuration
    - planning-for-availability
2. Size your system. For more information, see design-decisions.
3. Install Payara Server
4. If you plan to administer your clusters centrally, set up secure shell (SSH) for centralized administration.
5. Configure domains, nodes, clusters/deployment groups, instances, and virtual servers as required.

    For more information, see the following documentation:

    - "administering-domains" in the Payara Server General Administration section
    - "administering-payara-server-nodes" in the Payara Server High Availability section
    - "administering-payara-server-clusters" in the Payara Server High Availability

section

- "administering-payara-server-instances" in the Payara Server High Availability section
- "administering-virtual-servers" in the Payara Server General Administration section

6. Configure your load balancer, tentatively, using Apache Mod JK. For more information, see "administering-mod_jk" in the Payara Server General Administration section.
7. Configure the web container and EJB container for replicated session persistence. For more information, see "configuring-high-availability-session-persistence-and-failover" in the Payara Server High Availability section.
8. If you are using messaging extensively, configure Java Message Service (JMS) clusters for failover.

   For more information, see the following documentation:

   - planning-message-queue-broker-deployment
   - "configuring-java-message-service-high-availability" in the Payara Server High Availability section

Planning Your Deployment

Before deploying Payara Server, first determine the performance and availability goals, and then make decisions about the hardware, network, and storage requirements accordingly.

Establishing Performance Goals

At its simplest, high performance means maximizing throughput and reducing response time. Beyond these basic goals, you can establish specific goals by determining the following:

- What types of applications and services are deployed, and how do clients access them?

- Which applications and services need to be highly available?
- Do the applications have session state or are they stateless?
- What request capacity or throughput must the system support?
- How many concurrent users must the system support?
- What is an acceptable average response time for user requests?
- What is the average think time between requests?

You can calculate some of these metrics using a remote browser emulator (RBE) tool, or website performance and benchmarking software that simulates expected application activity. Typically, RBE and benchmarking products generate concurrent HTTP requests and then report the response time for a given number of requests per minute. You can then use these figures to calculate server activity.

The results of the calculations described in this chapter are not absolute. Treat them as reference points to work against, as you fine-tune the performance of Payara Server and your applications.

Examples of RBEs include: Puppeteer, Apache JMeter, LoadRunner, Selenium etc.

Estimating Throughput

In broad terms, throughput measures the amount of work performed by Payara Server. For Payara Server, throughput can be defined as the number of requests processed per minute per server instance.

As described in the next section, Payara Server throughput is a function of many factors, including the nature and size of user requests, number of users, performance of Payara Server instances and back-end databases. You can estimate throughput on a single machine by benchmarking with simulated workloads.

High availability applications incur additional overhead because they periodically save session data. The amount of overhead depends on the amount of data, how frequently it changes, and how often it is saved. The first two factors depend on the application in

question; the latter is also affected by server settings.

Estimating Load on Payara Server Instances

Consider the following factors to estimate the load on Payara Server instances.

Maximum Number of Concurrent Users

Users interact with an application through a client, such as a web browser or Java program. Based on the user's actions, the client periodically sends requests to the Payara Server. A user is considered active as long as the user's session has neither expired nor been terminated. When estimating the number of concurrent users, include all active users.

Initially, as the number of users increases, throughput increases correspondingly. However, as the number of concurrent requests increases, server performance begins to saturate, and throughput begins to decline.

Identify the point at which adding concurrent users reduces the number of requests that can be processed per minute. This point indicates when optimal performance is reached and beyond which throughput starts to degrade. Generally, strive to operate the system at optimal throughput as much as possible. You might need to add processing power to handle additional load and increase throughput.

Think Time

A user does not submit requests continuously. A user submits a request, the server receives and processes the request, and then returns a result, at which point the user spends some time before submitting a new request. The time between one request and the next is called think time.

Think times are dependent on the type of users. For example, machine-to-machine interaction such as for a web service typically has a lower think time than that of a human user. You may have to consider a mix of machine and human interactions to estimate think time.

Determining the average think time is important. You can use this duration to calculate

the number of requests that need to be completed per minute, as well as the number of concurrent users the system can support.

Average Response Time

Response time refers to the amount of time Payara Server takes to return the results of a request to the user. The response time is affected by factors such as network bandwidth, number of users, number and type of requests submitted, and the average think time.

In this section, response time refers to the mean, or average, response time. Each type of request has its own minimal response time. However, when evaluating system performance, base the analysis on the average response time of all requests.

The faster the response time, the more requests per minute are being processed. However, as the number of users on the system increases, the response time starts to increase as well, even though the number of requests per minute declines.

A system performance graph indicates that after a certain point, requests per minute are inversely proportional to response time. The sharper the decline in requests per minute, the steeper the increase in response time.

The point of the peak load is the point at which requests per minute start to decline. Prior to this point, response time calculations are not necessarily accurate because they do not use peak numbers in the formula. After this point, (because of the inversely proportional relationship between requests per minute and response time), the administrator can more accurately calculate response time using maximum number of users and requests per minute.

Use the following formula to determine $T_{response}$, the response time (in seconds) at peak load:

$$T_{response} = n/r - T_{think}$$

where

- `n` is the number of concurrent users
- `r` is the number requests per second the server receives
- T_{think} is the average think time (in seconds) to obtain an accurate response time result, always include think time in the equation.

Calculation of Response Time

Consider the following example:

- Assume maximum number of concurrent users, n, is 5,000.
- Assume maximum number of requests, r, 1,000 per second.
- Assume average think time, T_{think}, is three seconds per request.

Thus, the calculation of response time is:

$T_{response} = n/r - T_{think} = (5000/ 1000) - 3 sec. = 5 - 3 sec.$

Therefore, the response time is two seconds.

After the system's response time has been calculated, particularly at peak load, compare it to the acceptable response time for the application. Response time, along with throughput, is one of the main factors critical to Payara Server performance.

Requests per Minute

If you know the number of concurrent users at any given time, the response time of their requests, and the average user think time, then you can calculate the number of requests per minute. Typically, start by estimating the number of concurrent users that are on the system.

For example, after running website performance software, the administrator concludes that the average number of concurrent users submitting requests on an online banking website is `3,000`. This number depends on the number of users who have signed up to be members of the online bank, their banking transaction behavior, the time of the day or week they choose to submit requests, and so on.

Therefore, knowing this information enables you to use the requests per minute formula described in this section to calculate how many requests per minute your system can handle for this user base. Since requests per minute and response time become inversely proportional at peak load, decide if fewer requests per minute is acceptable as a trade-off for better response time, or alternatively, if a slower response time is acceptable as a trade-off for more requests per minute.

Experiment with the requests per minute and response time thresholds that are acceptable as a starting point for fine-tuning system performance. Thereafter, decide which areas of the system require adjustment.

Solving for `r` in the equation in the previous section gives:

$r = n/(T_{response} + T_{think})$

Calculation of Requests Per Second

Consider the following example:

- n = 2,800 concurrent users
- $T_{response}$ = 1 (one second per request average response time)
- T_{think} = 3, (three seconds average think time)

The calculation for the number of requests per second is:

```
r = 2800 / (1+3) = 700
```

Therefore, the number of requests per second is 700 and the number of requests per minute is 42000.

Planning the Network Configuration

When planning how to integrate the Payara Server into the network, estimate the bandwidth requirements and plan the network in such a way that it can meet users' performance requirements.

Estimating Bandwidth Requirements

To decide on the desired size and bandwidth of the network, first determine the network traffic and identify its peak. Check if there is a particular hour, day of the week, or day of the month when overall volume peaks, and then determine the duration of that peak.

During peak load times, the number of packets in the network is at its highest level. In general, if you design for peak load, scale your system with the goal of handling 100 percent of peak volume. Bear in mind, however, that any network behaves unpredictably and that despite your scaling efforts, it might not always be able to handle 100 percent of peak volume.

For example, assume that at peak load, five percent of users occasionally do not have immediate network access when accessing applications deployed on Payara Server. Of that five percent, estimate how many users retry access after the first attempt. Again, not all of those users might get through, and of that unsuccessful portion, another percentage will retry. As a result, the peak appears longer because peak use is spread out over time as users continue to attempt access.

Calculating Bandwidth Required

Based on the calculations made in establishing-performance-goals, determine the additional bandwidth required for deploying Payara Server at your site.

Depending on the method of access (T-1 lines, ADSL, cable modem, and so on), calculate the amount of increased bandwidth required to handle your estimated load. For example, suppose your site uses T-1 or higher-speed T-3 lines. Given their bandwidth, estimate how many lines are needed on the network, based on the average number of requests generated per second at your site and the maximum peak load. Calculate these figures using a website analysis and monitoring tool.

Calculation of Bandwidth Required

A single T-1 line can handle 1.544 Mbps. Therefore, a network of four T-1 lines can handle approximately 6 Mbps of data. Assuming that the average HTML page sent back to a client is 30 kilobytes (KB), this network of four T-1 lines can handle the following

traffic per second:

6,176,000 bits/10 bits = 772,000 bytes per second

772,000 bytes per second/30 KB = approximately 25 concurrent response pages per second.

With traffic of 25 pages per second, this system can handle 90,000 pages per hour (25 x 60 seconds x 60 minutes), and therefore 2,160,000 pages per day maximum, assuming an even load throughout the day. If the maximum peak load is greater than this, increase the bandwidth accordingly.

Estimating Peak Load

Having an even load throughout the day is probably not realistic. You need to determine when the peak load occurs, how long it lasts, and what percentage of the total load is the peak load.

Calculation of Peak Load

If the peak load lasts for two hours and takes up 30 percent of the total load of 2,160,000 pages, this implies that 648,000 pages must be carried over the T-1 lines during two hours of the day.

Therefore, to accommodate peak load during those two hours, increase the number of T-1 lines according to the following calculations:

648,000 pages/120 minutes = 5,400 pages per minute

5,400 pages per minute/60 seconds = 90 pages per second

If four lines can handle 25 pages per second, then approximately four times that many pages requires four times that many lines, in this case 16 lines. The 16 lines are meant for handling the realistic maximum of a 30 percent peak load. Obviously, the other 70 percent of the load can be handled throughout the rest of the day by these many lines.

Rightsizing Availability

To plan availability of systems and applications, assess the availability needs of the user groups that access different applications. For example, external fee-paying users and business partners often have higher quality of service (QoS) expectations than internal users. Thus, it may be more acceptable to internal users for an application feature, application, or server to be unavailable than it would be for paying external customers.

There is an increasing cost and complexity to mitigating against decreasingly probable events. At one end of the continuum, a simple load-balanced cluster can tolerate localized application, middleware, and hardware failures. At the other end of the scale, geographically distinct clusters can mitigate against major catastrophes affecting the entire data center.

To realize a good return on investment, it often makes sense to identify availability requirements of features within an application. For example, it may not be acceptable for an insurance quotation system to be unavailable (potentially turning away new business), but brief unavailability of the account management function (where existing customers can view their current coverage) is unlikely to turn away existing customers.

Using Clusters to Improve Availability

At the most basic level, a cluster is a group of Payara Server clients as a single instance. This provides horizontal scalability as well as higher availability than a single instance on a single machine. The ORB and integrated JMS brokers also perform load balancing to Payara Server clusters. If an instance fails, becomes unavailable (due to network faults), or becomes unresponsive, requests are redirected only to existing, available machines.

Adding Redundancy to the System

One way to achieve high availability is to add hardware and software redundancy to the system. When one unit fails, the redundant unit takes over. This is also referred to as fault tolerance. In general, to maximize high availability, determine and remove every

possible point of failure in the system.

Identifying Failure Classes

The level of redundancy is determined by the failure classes (types of failure) that the system needs to tolerate. Some examples of failure classes are:

- System process
- Machine
- Power supply
- Disk
- Network failures
- Building fires or other preventable disasters
- Unpredictable natural catastrophes

Duplicated system processes tolerate single system process failures, as well as single machine failures. Attaching the duplicated mirrored (paired) machines to different power supplies tolerates single power failures. By keeping the mirrored machines in separate buildings, a single-building fire can be tolerated. By keeping them in separate geographical locations, natural catastrophes like earthquakes can be tolerated.

Planning Failover Capacity

Failover capacity planning implies deciding how many additional servers and processes you need to add to the Payara Server deployment so that in the event of a server or process failure, the system can seamlessly recover data and continue processing. If your system gets overloaded, a process or server failure might result, causing response time degradation or even total loss of service. Preparing for such an occurrence is critical to successful deployment.

To maintain capacity, especially at peak loads, add spare machines running Payara Server instances to the existing deployment.

For example, consider a system with two machines running one Payara Server instance

each. Together, these machines handle a peak load of 300 requests per second. If one of these machines becomes unavailable, the system will be able to handle only 150 requests, assuming an even load distribution between the machines. Therefore, half the requests during peak load will not be served.

Design Decisions

Design decisions include whether you are designing the system for peak or steady-state load, the number of machines in various roles and their sizes, and the size of the administration thread pool.

Designing for Peak or Steady State Load

In a typical deployment, there is a difference between steady state and peak workloads:

- If the system is designed to handle peak load, it can sustain the expected maximum load of users and requests without degrading response time. This implies that the system can handle extreme cases of expected system load. If the difference between peak load and steady state load is substantial, designing for peak loads can mean spending money on resources that are often idle.
- If the system is designed to handle steady state load, it does not have all the resources required to handle the expected peak load. Thus, the system has a slower response time when peak load occurs.

How often the system is expected to handle peak load will determine whether you want to design for peak load or for steady state.

If peak load occurs often (let's say, several times per day), it may be worthwhile to expand capacity to handle it. If the system operates at steady state 90 percent of the time, and at peak only 10 percent of the time, then it may be preferable to deploy a system designed around steady state load. This implies that the system's response time will be slower only 10 percent of the time. Decide if the frequency or duration of time that the system operates at peak justifies the need to add resources to the system.

System Sizing

Based on the load on the Payara Server instances and failover requirements, you can determine the number of applications server instances (hosts) needed. Evaluate your environment on the basis of the factors explained in estimating-load-on-payara-server-instances to each Payara Server instance, although each instance can use more than one Central Processing Unit (CPU).

Sizing the Administration Thread Pool

The default `admin-thread-pool` size of 50 should be adequate for most cluster deployments. If you have unusually large clusters, you may need to increase this thread pool size. In this case, set the `max-thread-pool-size` attribute to the number of instances in your largest cluster, but not larger than the number of incoming synchronization requests that the DAS can handle.

Planning Message Queue Broker Deployment

The Jakarta Messaging (JMS) API is a messaging standard that allows Jakarta EE applications and components to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous. Message Queue, which implements JMS, is integrated with Payara Server, enabling you to create components that send and receive JMS messages, including message-driven beans (MDBs).

Message Queue is integrated with Payara Server using a resource adapter also known as a connector module. A resource adapter is a Jakarta EE component defined according to the Jakarta Connectors (JCA) Specification. This specification defines a standardized way in which application servers such as Payara Server can integrate with enterprise information systems such as JMS providers. Payara Server includes a resource adapter that integrates with its own JMS provider, Message Queue. To use a different JMS provider, you must obtain and deploy a suitable resource adapter that is designed to integrate with it.

Creating a JMS resource in Payara Server using the Administration Console creates a reconfigured connector resource that uses the Message Queue resource adapter. To

create JMS Resources that use any other resource adapter (including `GenericJMSRA`), you must create them under the Connectors node in the Administration Console.

In addition to using resource adapter APIs, Payara Server uses additional Message Queue APIs to provide better integration with Message Queue. This tight integration enables features such as connector failover, load balancing of outbound connections, and load balancing of inbound messages to MDBs. These features enable you to make messaging traffic fault-tolerant and highly available.

Multi-Broker Clusters

Message Queue supports using multiple interconnected broker instances known as a broker cluster. With broker clusters, client connections are distributed across all the brokers in the cluster. Clustering provides horizontal scalability and improves availability.

A single message broker scales to about eight CPUs and provides sufficient throughput for typical applications. If a broker process fails, it is automatically restarted. However, as the number of clients connected to a broker increases, and as the number of messages being delivered increases, a broker will eventually exceed limitations such as number of file descriptors and memory.

Having multiple brokers in a cluster rather than a single broker enables you to:

- Provide messaging services despite hardware failures on a single machine.
- Minimize downtime while performing system maintenance.
- Accommodate workgroups having different user repositories.
- Deal with firewall restrictions.

Message Queue allows you to create conventional or enhanced broker clusters. Conventional broker clusters offer service availability. Enhanced broker clusters offer both service and data availability.

In a conventional cluster, having multiple brokers does not ensure that transactions in

progress at the time of a broker failure will continue on the alternate broker. Although Message Queue reestablishes a failed connection with a different broker in a cluster, transactions owned by the failed broker are not available until it restarts. Except for failed in-progress transactions, user applications can continue on the failed-over connection. Service failover is thus ensured.

In an enhanced cluster, transactions and persistent messages owned by the failed broker are taken over by another running broker in the cluster and non-prepared transactions are rolled back. Data failover is ensured for prepared transactions and persisted messages.

Master Broker and Client Synchronization for Conventional Clusters

In a configuration for a conventional broker cluster, each destination is replicated on all the brokers in a cluster. Each broker knows about message consumers that are registered for destinations on all other brokers. Each broker can therefore route messages from its own directly-connected message producers to remote message consumers, and deliver messages from remote producers to its own directly-connected consumers.

In a cluster configuration, the broker to which each message producer is directly connected performs the routing for messages sent to it by that producer. Hence, a persistent message is both stored and routed by the message's home broker.

Whenever an administrator creates or destroys a destination on a broker, this information is automatically propagated to all other brokers in a cluster. Similarly, whenever a message consumer is registered with its home broker, or whenever a consumer is disconnected from its home broker—either explicitly or because of a client or network failure, or because its home broker goes down—the relevant information about the consumer is propagated throughout the cluster. In a similar fashion, information about durable subscriptions is also propagated to all brokers in a cluster.

A shared database of cluster change records can be configured as an alternative to using a master broker. For more information, "using-message-queue-broker-clusters-

with-payara-server" in the Payara Server High Availability section.

Configuring Payara Server to Use Message Queue Brokers

By default, Message Queue brokers (JMS hosts) run in the same JVM as the Payara Server process. However, Message Queue brokers (JMS hosts) can be configured to run in a separate JVM from the Payara Server process. This allows multiple Payara Server instances or clusters to share the same set of Message Queue brokers.

The Payara Server's Java Message Service represents the connector module (resource adapter) for Message Queue. You can manage the Java Message Service through the Administration Console or the `asadmin` command-line utility.

In Payara Server, a JMS host refers to a Message Queue broker. The Payara Server's Java Message Service configuration contains a JMS Host List (also called AddressList) that contains all the JMS hosts that will be used.

Java Message Service Type

There are three types of integration between Payara Server and Message Queue brokers: `embedded`, `local`, and `remote`. You can set this type attribute on the Administration Console's Java Message Service page.

Embedded Java Message Service

If the Type attribute is `EMBEDDED`, Payara Server and the JMS broker are co-located in the same virtual machine. The JMS Service is started in-process and managed by Payara Server. In `EMBEDDED` mode, JMS operations on stand-alone server instances bypass the networking stack, which leads to performance optimization. The `EMBEDDED` type is most suitable for stand-alone Payara Server instances. `EMBEDDED` mode is not supported for enhanced broker clusters and not recommended for production environments.

With the `EMBEDDED` type, use the **Start Arguments** attribute to specify Message Queue broker startup parameters.

With the `EMBEDDED` type, make sure the Java heap size is large enough to allow Payara

Server and Message Queue to run in the same virtual machine.

Local Java Message Service

If the Type attribute is `LOCAL`, Payara Server starts and stops the Message Queue broker. When Payara Server starts up, it starts the Message Queue broker specified as the Default JMS host. Likewise, when the Payara Server instance shuts down, it shuts down the Message Queue broker. The `LOCAL` type is most suitable for use with enhanced broker clusters, and for other cases where the administrator prefers the use of separate JVMs.

With the `LOCAL` type, use the **Start Arguments** attribute to specify Message Queue broker startup parameters.

Remote Java Message Service

If the Type attribute is `REMOTE`, Payara Server uses an externally configured broker or broker cluster. In this case, you must start and stop Message Queue brokers separately from Payara Server, and use Message Queue tools to configure and tune the broker or broker cluster. The `REMOTE` type is most suitable for brokers running on different machines from the server instances (to share the load among more machines or for higher availability), or for using a different number of brokers and server instances.

With the `REMOTE` type, you must specify Message Queue broker startup parameters using the Message Queue tools. The **Start Arguments** attribute is ignored.

Managing JMS with the Administration Console

In the Administration Console, you can set JMS properties using the Java Message Service node for a particular configuration. You can set properties such as **Reconnect Interval** and **Reconnect Attempts**. For more information, see "administering-the-java-message-service-jms" in the Payara Server General Administration section.

The JMS Hosts node under the Java Message Service node contains a list of JMS hosts. You can add and remove hosts from the list. For each host, you can set the host name, port number, and the administration username and password. By default, the JMS Hosts list contains one Message Queue broker, called `default_JMS_host`, that

represents the local Message Queue broker integrated with Payara Server.

In `REMOTE` mode, configure the JMS Hosts list to contain all the Message Queue brokers in the cluster. For example, to set up a cluster containing three Message Queue brokers, add a JMS host within the Java Message Service for each one. Message Queue clients use the configuration information in the Jakarta Messaging system to communicate with the Message Queue broker.

Managing JMS with Asadmin

In addition to the Administration Console, you can use the `asadmin` command-line utility to manage the Java Message Service and JMS hosts. Use the following `asadmin` commands:

- Configuring Java Message Service attributes: `asadmin set`
- Managing JMS hosts:
  - `asadmin create-jms-host`
  - `asadmin delete-jms-host`
  - `asadmin list-jms-hosts`
- Managing JMS resources:
  - `asadmin create-jms-resource`
  - `asadmin delete-jms-resource`
  - `asadmin list-jms-resources`

Default JMS Host

You can specify the default JMS Host in the Administration Console Java Message Service page. If the Java Message Service type is `LOCAL`, Payara Server starts the default JMS host when the Payara Server instance starts. If the Java Message Service type is `EMBEDDED`, the default JMS host is started lazily when needed.

In `REMOTE` mode, to use a Message Queue broker cluster, delete the default JMS host, then add all the Message Queue brokers in the cluster as JMS hosts. In this case, the

default JMS host becomes the first JMS host in the JMS host list.

You can also explicitly set the default JMS host to one of the JMS hosts. When the Payara Server uses a Message Queue cluster, the default JMS host executes Message Queue-specific commands. For example, when a physical destination is created for a Message Queue broker cluster, the default JMS host executes the command to create the physical destinations, but all brokers in the cluster use the physical destination.

Example Deployment Scenarios

To accommodate your messaging needs, modify the Java Message Service and JMS host list to suit your deployment, performance, and availability needs. The following sections describe some typical scenarios.

For best availability, deploy Message Queue brokers and Payara Servers on different machines, if messaging needs are not just with Payara Server. Another option is to run a Payara Server instance and a Message Queue broker instance on each machine until there is sufficient messaging capacity.

Default Deployment

Installing the Payara Server automatically creates a domain administration server (DAS). By default, the Java Message Service type for the DAS is `EMBEDDED`. So, starting the DAS also starts its default Message Queue broker.

Creating a new domain also creates a new broker. By default, when you add a stand-alone server instance or a cluster to the domain, its Java Message Service is configured as `EMBEDDED` and its default JMS host is the broker started by the DAS.

Using a Message Queue Broker Cluster with a Payara Server Cluster

In `EMBEDDED` or `LOCAL` mode, when a Payara Server is configured, a Message Queue broker cluster is autoconfigured with each Payara Server instance associated with a Message Queue broker instance.

In `REMOTE` mode, to configure a Payara Server cluster to use a Message Queue broker cluster, add all the Message Queue brokers as JMS hosts in the Payara Server's Java

Message Service. Any JMS connection factories created and MDBs deployed then uses the JMS configuration specified.

Specifying an Application-Specific Message Queue Broker Cluster

In some cases, an application may need to use a different Message Queue broker cluster than the one used by the Payara Server cluster. To do so, use the `AddressList` property of a JMS connection factory or the `activation-config` element in an MDB deployment descriptor to specify the Message Queue broker cluster.

For more information about configuring connection factories, see "administering-jms-connection-factories-and-destinations" in the Payara Server Administration section. For more information about MDBs, see "using-message-driven-beans" in the Payara Server Application Development section.

Application Clients

When an application client or standalone application accesses a JMS administered object for the first time, the client JVM retrieves the Java Message Service configuration from the server. Furthermore, changes to the JMS service will not be available to the client JVM until it is restarted.

Deployment Checklist

This appendix provides a checklist to get started on evaluation and production using Payara Server.

Checklist

This table provides a checklist of the steps to be performed before deploying an application to production.

Table 16. Checklist

Component/Feature	Description
Application	Determine the following requirements for the application to be deployed. • Required/acceptable response time. • Peak load characteristics. • Necessary persistence scope and frequency. • Session timeout in `web.xml`. • Fail-over and availability requirements.
Hardware	• Have necessary amounts of hard disk space and memory installed. • Use the sizing exercise to identify the requirements for deployment.
Operating System	• Ensure that the product is installed on a supported platform. • Ensure that the patch levels are up-to-date and accurate.
Network Infrastructure	• Identify single points of failures and address them. • Make sure that the NICs and other network components are correctly configured. • Run `ttcp` benchmark test to determine if the throughput meets the requirements/expected result. • Setup `ssh` based on your preference.
Back-ends and other external data sources	Check with the domain expert or vendor to ensure that these data sources are configured appropriately.
System Changes/Configuration	• Make sure that changes to `/etc/system` and its equivalent on Windows are completed before running any performance/stress tests. • Make sure the changes to the TCP/IP settings are complete. • By default, the system comes with lots of services pre-configured. Not all of them are required to be running. Turn off services that are not needed to conserve system resources.

Component/Feature	Description
Installation	• Ensure that these servers are not installed on NFS mounted volumes. • Check for enough disk space and RAM when installing Payara Server.
Payara Server Configuration	• Logging: Enable access log rotation. • Choose the right logging level. `WARNING` is usually appropriate. • Configure Jakarta EE containers using the Administration Console. • Configure HTTP listeners using the Administration Console. • Configure ORB thread pool using the Administration Console. • Ensure that the appropriate persistence scope and frequency are used, and they are not overridden underneath in the individual Web/EJB modules. • Ensure that only critical methods in the SFSB are checkpointed.
Java Virtual Machine Configuration	• Initially set the minimum and maximum heap sizes to be the same, and at least one GB for each instance. • See Java Hotspot VM Options for more information. • When running multiple instances of Payara Server, consider creating a processor set and bind Payara Server to it. This helps in cases where the CMS collector is used to sweep the old generation.
Configuring time-outs in Payara Server	**`max-wait-time-millis`**: Wait time to get a connection from the pool before throwing an exception. Default is 6s. Consider changing this value for highly loaded systems where the size of the data being persisted is greater than 50 KB. **`cache-idle-timeout-in-seconds`**: Time an EJB is allowed to be idle in the cache before it gets passivated. Applies only to entity beans and stateful session beans. **`removal-timeout-in-seconds`**: Time that an EJB remains passivated (idle in the backup store). Default value is 60 minutes. Adjust this value based on the need for SFSB failover.

Component/Feature	Description
Tune VM Garbage Collection (GC)	Garbage collection pauses of four seconds or more can cause intermittent problems in persisting session state. To avoid this problem, tune the VM heap. In cases where even a single failure to persist data is unacceptable or when the system is not fully loaded, use the CMS collector or the throughput collector. These can be enabled by adding: `<jvm-options>-XX:+UseConcMarkSweepGC</jvm-options>` This option may decrease throughput.

Application Deployment

Overview of Payara Server Application Deployment

Payara Server provides an environment for developing and deploying Jakarta applications and web services. Payara Server applications include Jakarta Platform, Enterprise Edition (Jakarta EE platform) standard features as well as features specific to Payara Server. This guide explains the tools and processes used for deploying applications and modules in the Payara Server environment. Only Payara Server features are described in detail in this document.

Information and instructions on deploying from the command line are provided in this document. Information and instructions for accomplishing the deployment tasks by using the Administration Console are contained in the Administration Console online help.

About Application Deployment

Assembly, also known as packaging, is the process of combining discrete components of an application or module into a single unit that can be installed on an application server. The Payara Server assembly process conforms to the customary Jakarta EE specifications. The only difference is that when you assemble applications or modules in Payara Server, you can include optional Payara Server deployment descriptors that enhance functionality.

Deployment is the process of installing an application or module on Payara Server, optionally specifying location-specific information, such as a list of local users that can access the application, or the name of the local database. Payara Server deployment tools expand the archive file into an open directory structure that is ready for users. Payara Server deployment tools are described in "About Deployment Tools".

The following topics are addressed here:

- "General Deployment Functionality"
- "Deployment Descriptors and Annotations"
- "Modules and Applications"
- "Access to Shared Framework Classes"
- "Naming Standards"
- "Module and Application Versions"

General Deployment Functionality

Various Jakarta EE module types, such as connector module, web module, EJB module, application client module, can be deployed in the following ways:

- Archive Deployment. Deploys the application as an archive file. For instructions, see to-deploy-an-application-or-module.
- Dynamic Reloading. Redeploys the application by creating or modifying a special `.reload` file in the **applications** repository. For instructions, see to-reload-changes-to-applications-or-modules-dynamically.
- Automatic Deployment. Deploys the application archive that is placed in the `autodeployment` directory. For instructions, see to-deploy-an-application-or-module-automatically.
- Directory Deployment. Deploys the application in a directory format. For instructions, see to-deploy-an-application-or-module-in-a-directory-format.

A deployment plan, which deploys a portable archive along with a deployment plan

containing Payara Server deployment descriptors, can apply to any of these deployment techniques. For instructions, see to-deploy-an-application-or-module-by-using-a-deployment-plan.

There are two work situations that require different safeguards and processes:

- A development environment provides a loose set of tools and work spaces for a relatively small number of developers who are creating and testing applications and modules.
- A production environment provides a stable, protected environment where applications are tuned to maximum efficiency for business use rather than for development.

Some deployment methods that are used effectively in a development environment should not be used in production. In addition, whenever a reload is done, the sessions that are in transit become invalid, which might not be a concern for development, but can be a serious matter in production. The client must restart the session, another negative in a production environment.

For production environments, any upgrade should be performed as a **rolling upgrade**, which upgrades applications and modules without interruption in service.

Deployment Descriptors and Annotations

A deployment descriptor is an XML file that describes how a Jakarta EE application or module should be deployed. Each deployment descriptor XML file has a corresponding Document Type Definition (DTD) file or schema (XSD) file, which defines the elements, data, and attributes that the deployment descriptor file can contain. The deployment descriptor directs a deployment tool to deploy a module or application with specific container options, and also describes specific configuration requirements that you must resolve.

Because the information in a deployment descriptor is declarative, it can be changed without requiring modifications to source code. During deployment, Payara Server reads

the information in the deployment descriptor and deploys the application or module as directed.

The following types of deployment descriptors are associated with Payara Server:

- **Jakarta EE Standard Descriptors**. Jakarta EE standard deployment descriptors are described in the Jakarta EE specification. You can find the specification at `https://jakarta.ee/specifications/`. Information about the XML schemas that define Jakarta EE standard deployment descriptors is available at `https://jakarta.ee/xml/ns/jakartaee/`.

  The Jakarta EE specification permits the use of alternate top-level standard deployment descriptors that reside outside the application archive using the `alt-dd` mechanism (alternate module-level deployment descriptors were permitted prior to Java EE 7). Alternate deployment descriptors are described in the Jakarta EE specification. You can find the specification at `https://jakarta.ee/specifications/`. Alternate deployment descriptors override the top-level deployment descriptors packaged in an application archive.

  For example, for EAR files, an alternate deployment descriptor overrides `application.xml`.

  For standalone modules, an alternate deployment descriptor overrides the top-level module descriptor, such as `web.xml`.

- **Payara Server Descriptors**. Payara Server provides optional deployment descriptors for configuring features that are specific to Payara Server. For example, when you assemble an EJB module, you annotate or create two Payara Server deployment descriptor files with these names: `ejb-jar.xml` and `glassfish-ejb-jar.xml`.

  For complete descriptions of these files and their elements, see about-the-payara-platform-deployment-descriptors and elements-of-the-payara-platform-deployment-descriptors.

  Payara Server also permits the use of alternate top-level Payara Server runtime

deployment descriptors that reside outside an application archive. Alternate Payara Server deployment descriptors override the top-level deployment descriptors packaged in the archive.

For example, for EAR files, an alternate Payara Server deployment descriptor overrides `payara-application.xml`. For standalone modules, an alternate Payara Server deployment descriptor overrides the top-level module descriptor, such as `payara-web.xml`. The name of the Payara Server alternate deployment descriptor file must begin with `payara-`.

Alternate deployment descriptors do not apply to `sun-*.xml` deployment descriptors.

Unless otherwise stated, settings in the Payara Server deployment descriptors override corresponding settings in the Jakarta EE standard descriptors and in the Payara Server configuration.

An annotation, also called metadata, enables a declarative style of programming. You can specify information within a class file by using annotations.

When the application or module is deployed, the information can either be used or overridden by the deployment descriptor. Payara Server supports annotation according to the following specifications:

- Jakarta Annotations
- Jakarta Web Services Metadata
- Jakarta Enterprise Beans

The following annotation and deployment descriptor combinations are supported:

- Jakarta EE applications or modules can be packaged with full Jakarta EE compliant standard and runtime deployment descriptors. If the standard deployment descriptors have specified the `metadata-complete` attribute, annotations in the application or module are ignored.

- Jakarta EE applications or modules can be fully annotated with metadata defined by the listed specifications. Annotation eliminates the need for Jakarta EE standard deployment descriptors. In most cases, the Payara Server deployment descriptors are also not needed.
- Jakarta EE applications or modules can be partially annotated with some deployment information in standard deployment descriptors. In case of conflicts, deployment descriptor values supersede the annotated metadata, and a warning message is logged.

Modules and Applications

An application is a logical collection of one or more modules joined by application annotations or deployment descriptors. You assemble components into JAR, WAR, or RAR files, then combine these files and, optionally, deployment descriptors into an Enterprise archive (EAR) file which is deployed.

A module is a collection of one or more Jakarta EE components that run in the same container type, such as a web container or EJB container. The module uses annotations or deployment descriptors of that container type. You can deploy a module alone or as part of an application.

The following topics are addressed here:

- "Types of Modules"
- "Module-Based Deployment"
- "Application-Based Deployment"

Types of Modules

Payara Server supports the following types of modules:

- **Web Module**. A web module, also known as a web application, is a collection of servlets, EJBs, HTML pages, classes, and other resources that you can bundle and deploy to several Jakarta EE application servers. A web application archive (WAR) file is the standard format for assembling web applications. A WAR file can consist of

the following items: servlets, Jakarta Server Pages (JSP) files, JSP tag libraries, utility classes, static pages, client-side applets, beans, bean classes, enterprise bean classes, plus annotations or web deployment descriptors (`web.xml` and `payara-web.xml`).

- **EJB Module**. An EJB module is a deployable software unit that consists of one or more enterprise beans, plus an EJB deployment descriptor. A Java archive (JAR) file is the standard format for assembling enterprise beans. An EJB JAR file contains the bean classes (home, remote, local, and implementation), all of the utility classes, and annotations or deployment descriptors (`ejb-jar.xml` and `glassfish-ejb-jar.xml`).
- **Connector Module**. A connector module, also known as a resource adapter module, is a deployable software unit that provides a portable way for EJB components to access foreign enterprise information system (EIS) data. A connector module consists of all Jakarta interfaces, classes, and native libraries for implementing a resource module, plus a resource deployment descriptor. A resource adapter archive (RAR) is the standard format for assembling connector modules. Each Payara Server connector has annotations or a deployment descriptor file (`ra.xml`).

  After deploying a Jakarta EE connector module, you must configure it as described in developing-connectors in the Payara Server Application Development section.

- **Application Client Module**. An application client module is a deployable software unit that consists of one or more classes, and application client deployment descriptors (`application-client.xml` and `payara-application-client.xml`). An application client JAR file applies to a Payara Server type of Jakarta EE client. An application client supports the standard Jakarta EE Application Client specifications.

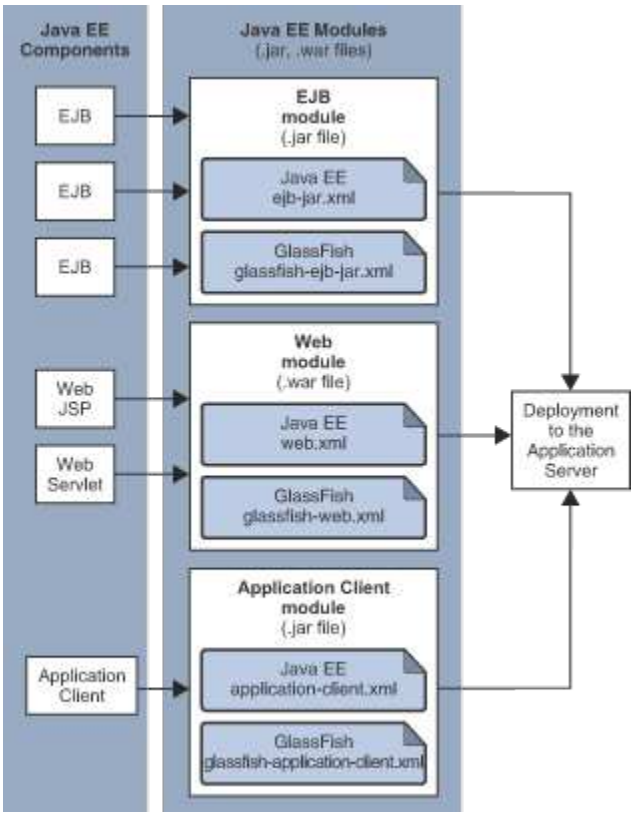
Module-Based Deployment

You can deploy web, EJB, and application client modules separately, outside any application. Module-based deployment is appropriate when components need to be accessed by other modules, applications, or application clients. Module-based deployment allows shared access to a bean from a web, EJB, or application client

component.

The following figure shows separately-deployed EJB, web, and application client modules.

Figure 1-1 Module-Based Assembly and Deployment

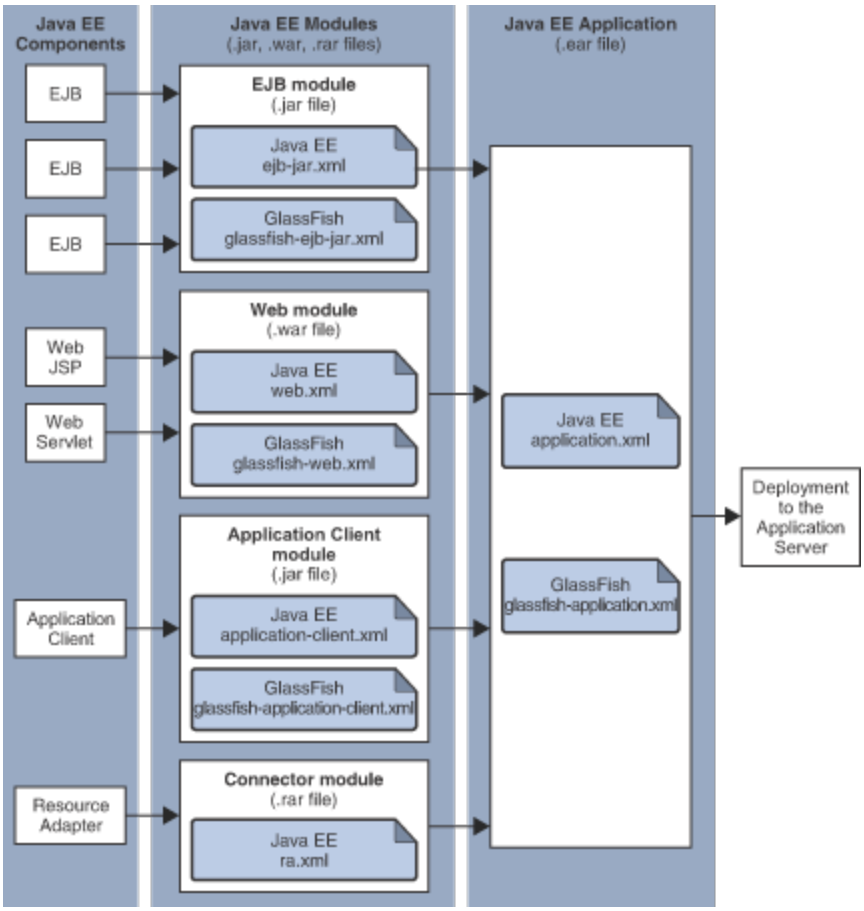


Application-Based Deployment

Application-based deployment is appropriate when components need to work together as one unit.

The following figure shows EJB, web, application client, and connector modules assembled into a Jakarta EE application.

Figure 1-2 Application-Based Assembly and Deployment



Access to Shared Framework Classes

If you assemble a large, shared library into every module that uses it, the result is a huge file that takes too long to register with the server. In addition, several versions of the same class could exist in different class loaders, which is a waste of resources. When Jakarta EE applications and modules use shared framework classes (such as utility classes and libraries), the classes can be put in the path for the common class loader or an application-specific class loader rather than in an application or module.

To specify an application-specific library file during deployment, use the `--libraries` option of the `deploy` or `redeploy` subcommand of the `asadmin` command. To add a library JAR file to the Common class loader directory, the Jakarta optional package directory, or the application-specific class loader directory, use the `add-library` subcommand. You can then list the libraries with `list-libraries` and remove the libraries with `remove-library`. For more information about all these commands, see the Payara Server Reference Manual.

For more information about class loaders, see classloading in the Payara Server Application Development section.

NOTE	According to the Jakarta EE specification, you cannot package utility classes within an individually-deployed EJB module. Instead, you must package the EJB module and utility JAR within an application using the JAR Extension Mechanism Architecture.

Naming Standards

Names of applications and individually-deployed modules must be unique within a Payara Server domain. Modules within an application must have unique names.

You should use a hierarchical naming scheme for module file names, EAR file names, module names as found in the `module-name` portion of the `ejb-jar.xml` files, and EJB names as found in the `ejb-name` portion of the `ejb-jar.xml` files. This hierarchical naming scheme ensures that name collisions do not occur. The benefits of this naming practice apply not only to Payara Server, but to other Jakarta EE application servers as well.

The following topics are addressed here:

- "Portable Naming"
- "JNDI Naming"
- "Directory Structure"

Portable Naming

The Jakarta EE specification defines the portable `application-name`, which allows you to specify an application name in the `application.xml` file.

For example:

```
<application-name>xyz</application-name>
```

The Jakarta EE specification also defines the portable `module-name` element in the

module standard deployment descriptors.

Payara Server determines the application registration name according to the following order of precedence:

1. The name specified at deployment time in the Administration Console or in the `--name` option of the `asadmin deploy` command is used.
2. If no name is specified at deployment time, the portable `application-name` or `module-name` in the Jakarta EE deployment descriptor is used.
3. If no name is specified at deployment time or in the deployment descriptors, the archive name, minus the file type suffix, is used.

Deploying from an IDE

When deploying an application or a module on Payara Server from an IDE (like **Apache NetBeans** or **IntelliJ Idea**), the IDE deploys to Payara Server using the asadmin deploy command, with the `--name` option specified.

This will force the application or module to have the specified name over the name defined in deployment descriptor. This is undesired because the IDE usually infers the application or module name from the name of the project or the package file and doesn't take the correct name of the application or module into account.

To solve the issue, application or module name defined in the deployment descriptor will **always** take precedence over the `--name` option. But you use the `--name` and `--forcename` options to override this behaviour.

JNDI Naming

Jakarta Naming and Directory Interface (JNDI) lookup names for EJB components must also be unique. Establishing a consistent naming convention can help. For example, appending the application name and the module name to the EJB name is a way to guarantee unique names, such as, `jms/qConnPool`.

Directory Structure

Application and module directory structures must follow the structure outlined in the Jakarta EE specification. During deployment, the application or module is expanded from the archive file to an open directory structure. The directories that hold the individual modules are named with `_jar`, `_rar`, and `_war` suffixes.

If you deploy a directory instead of an EAR file, your directory structure must follow this same convention. For instructions on performing directory deployment, see to-deploy-an-application-or-module-in-a-directory-format.

Module and Application Versions

Application and module versioning allows multiple versions of the same application to exist in a Payara Server domain, which simplifies upgrade and rollback tasks. At most one version of an application or module can be enabled on a server any given time. Versioning provides extensions to tools for deploying, viewing, and managing multiple versions of modules and applications, including the Administration Console and deployment-related `asadmin` subcommands. Different versions of the same module or application can have the same context root or JNDI name. Use of versioning is optional.

The following topics are addressed here:

- "Version Identifiers and Expressions"
- "Choosing the Enabled Version"
- "Versioning Restrictions and Limitations"

Version Identifiers and Expressions

The version identifier is a suffix to the module or application name. It is separated from the name by a colon (`:`). It must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. The following examples show valid version identifiers for the `foo` application:

```
foo:1
foo:BETA-2e
foo:3.8
```

```
foo:patch39875
```

A module or application without a version identifier is called the untagged version. This version can coexist with other versions of the same module or application that have version identifiers.

In some deployment-related `asadmin` commands, you can use an asterisk (`*`) as a wildcard character to specify a version expression, which selects multiple version identifiers. Using the asterisk by itself after the colon selects all versions of a module or application, including the untagged version. The following table shows example version expressions and the versions they select.

Version Expression	Selected Versions
`foo:*`	All versions of `foo`, including the untagged version
`foo:BETA*`	All `BETA` versions of `foo`
`foo:3.*`	All `3.`x versions of `foo`
`foo:patch*`	All `patch` versions of `foo`

The following table summarizes which `asadmin` subcommands are identifier-aware or expression-aware. All expression-aware subcommands are also identifier-aware.

Identifier-Aware Subcommands	Expression-Aware Subcommands
`deploy`, `redeploy`	`undeploy`
`enable`	`disable`
`list-sub-components`	`show-component-status`
`get-client-stubs`	`create-application-ref`, `delete-application-ref`

The `create-application-ref` subcommand is expression-aware only if the `--enabled` option is set to `false`. Because the `--enabled` option is set to `true` by default, the create-application-ref` subcommand is identifier-aware by default.

The `list-applications` and `list-application-refs` subcommands display

information about all deployed versions of a module or application. To find out which version is enabled, use the `--long` option.

Choosing the Enabled Version

At most one version of a module or application can be enabled on a server instance. All other versions are disabled. Enabling one version automatically disables all others. You can disable all versions of a module or application, leaving none enabled.

The `--enabled` option of the `deploy` and `redeploy` subcommands is set to `true` by default. Therefore, simply deploying or redeploying a module or application with a new version identifier enables the new version and disables all others. To deploy a new version in a disabled state, set the `--enabled` option to `false`.

To enable a version that has been deployed previously, use the `enable` subcommand.

Versioning Restrictions and Limitations

Module and application versioning in Payara Server is subject to the following restrictions and limitations:

- Use of the `--name` option is mandatory for modules and applications that use versioning. There is no automatic version identifier generation.
- Payara Server does not recognize any relationship between versions such as previous or later versions. All version relationships must be tracked manually.
- There is no limit to the number of versions you can deploy except what is imposed by disk space limits.
- A module or application in a directory should not be deployed twice with a different version identifier. To redeploy a module or application from a directory with a new version, you must use the `--force` option of the `deploy` subcommand.
- Database tables created or deleted as part of deployment and un-deployment are global resources and cannot be qualified by an application version. Be very careful when using global resources among versions of the same application.

- Web sessions are preserved during redeployment of a new version. However, preserving sessions among different versions of the same module or application is complex, because the key used for session variables is the same for the old and new versions.
- Resources are created with reference to a resource-adapter's module or application name. This means that an older version's resources do not automatically refer to a newer version of the module or application. Therefore, you must explicitly create resources for a newer version of a module or application. Payara Server ignores duplicate exported global resources and lets deployment succeed.
- OSGi already has its own versioning system. Therefore, when you deploy an OSGi bundle, Payara Server ignores any version information provided with the name but permits the deployment to succeed with warnings.

About Assembly and Deployment Events

The deployment tools that are provided by Payara Server can be used by any user authorized as an administrator to deploy applications and modules into any Payara Server environment. However, effective application deployment requires planning and care. Only the developer knows exactly what is required by an application, so the developer is responsible for initial assembly and deployment.

1. **Deployment Descriptor or Annotation Creation**. The developer creates the deployment descriptors or equivalent annotations using Jakarta standards and tools.

   Details of the Payara Server deployment descriptors are contained in about-the-payara-platform-deployment-descriptors and elements-of-the-payara-platform-deployment-descriptors.

   The Payara Server sample applications contain deployment descriptors that can be used as templates for developing deployment descriptors.

2. **Assembly**. The developer assembles the archive file(s) using Jakarta standards and tools, such as the `jar` command. The application or module is packaged into a JAR, WAR, RAR, or EAR file. For guidelines on naming, see naming-standards. There are no

Payara Server issues to consider.

3. **Test Deployment**. The developer performs a test deployment of the archive. For instructions, see to-deploy-an-application-or-module.
4. **Archive Submission**. The developer submits the verified archive to the administrator for deployment into a production environment. The developer includes instructions for any additional deployment tasks that the administrator must perform. For an example of such additional instructions, see access-to-shared-framework-classes.
5. **Configuration**. The administrator applies additional deployment specifics. Sometimes the developer has indicated additional deployment needs, such as specifying the production database. In this case, the administrator edits and reassembles the archive.
6. **Production Deployment**. The administrator deploys the archive to production. See to-deploy-an-application-or-module.
7. **Troubleshooting**. If deployment fails, the administrator returns the archive to the developer.

   The developer fixes the problem and resubmits the archive to the administrator. Sometimes the administrator resolves the problem, depending on what the problem is.

About Deployment Tools

Payara Server provides tools for assembling and deploying a module or application.

Administration Console

The Payara Server Administration Console is a browser-based utility that features a graphical interface that includes extensive online help for the administrative tasks. The format for starting the Administration Console in a web browser is `http://hostname:port`. For example:

```
http://localhost:4848
```

Step-by-step instructions for using the Administration Console for deployment are

provided in the Administration Console online help. You can display the help material for a page by clicking the Help button. The initial help page describes the functions and fields of the page itself. To find instructions for performing associated tasks, click a link in the See Also list.

The `asadmin` Utility

The Payara Server `asadmin` utility is a command-line tool that invokes subcommands for identifying the operation or task that you want to perform. You can run `asadmin` commands either from a command prompt or from a script. The format for starting the `asadmin` utility on the command line is ***`as-install`***`/bin/asadmin` ***`subcommand`*** ***`--option`***. For example:

```
asadmin list-applications --type web
```

Application deployment commands are listed in the-asadmin-deployment-subcommands.

For the most part, you can perform the same administrative tasks by using either the graphical Administration Console or the `asadmin` command-line utility, however, there are exceptions. Procedures for using the command-line utilities are provided in this guide and in the command-line help pages, which are similar to man pages. You can display the help material for a command by typing help followed by the subcommand. For example:

```
asadmin help list-applications
```

For additional information on the `asadmin` utility, see "using-the-asadmin-utility" in *Payara Server General Administration section* and the "`asadmin`(1M)" help page.

Additional Information on Application Deployment

As specified from Jakarta Platform, the following are relevant specifications to this guide:

- Jakarta EE Platform, Enterprise Edition Specification

`https://jakarta.ee/specifications/platform/`

- Jakarta Annotations Specification

  `https://jakarta.ee/specifications/annotations/`

- Jakarta Servlet Specification

  `https://jakarta.ee/specifications/servlet/`

- Jakarta Enterprise Beans Specification

  `https://jakarta.ee/specifications/enterprise-beans/`

- Jakarta Connectors Specification

  `https://jakarta.ee/specifications/connectors/`

Deploying Applications

This chapter provides procedures and guidelines for deploying applications and modules in the Payara Server environment by using the `asadmin`command-line utility.

Deploying Applications and Modules

Application deployment is a dynamic process, which means that deployed applications and modules become available without requiring you to restart the server instance.

Dynamic deployment can be useful in production environments to bring new applications and modules online easily. If you do restart the server, all deployed components are still deployed and available.

The following topics are addressed here:

- "To Deploy an Application or Module"
- "To Change Targets for a Deployed application or Module"
- "To List Deployed Applications or Modules"
- "To Redeploy an Application or Module"
- "To Disable an Application or Module"
- "To Enable an Application or Module"

- "To Undeploy an Application or Module"
- "To Reload Changes to Applications or Modules Dynamically"
- "To Deploy an Application or Module Automatically"
- "To Deploy an Application or Module by Using a Deployment Plan"
- "To Deploy an Application or Module in a Directory Format"
- "To Deploy an Application on Server start"
- "To Deploy from a URL"
- "To Deploy using a Maven Artifact"

Instructions for accomplishing these tasks by using the Administration Console are contained in the Administration Console online help.

To Deploy an Application or Module

Use the `deploy` subcommand in remote mode to deploy an assembled application or module to Payara Server. If an error occurs during deployment, the application or module is not deployed. If a module within an application contains an error, the entire application is not deployed.

These failures prevent a partial deployment that could leave the server in an inconsistent state.

By default, the deployment target is the default server instance, `server`. To deploy only to the default server instance, specify no target. If you deploy the application or module only to the domain target, it exists in the domain central repository, but no server instances or clusters can reference the component unless you add references.

You can also deploy a component to a specific stand-alone server instance or cluster. When you deploy to server instances or clusters, the application or module exists in the domain's central repository and is referenced by any clusters or server instances that you deployed to. For a cluster, the preselected deployment target is `server`.

If the component is already deployed or already exists, you can forcefully redeploy if you

set the `--force` option of the `deploy` subcommand to true. The `redeploy` subcommand also accomplishes this. See "this example".

You can see the enabled or disabled status of an application or module by using the "`show-component-status`" subcommand.

For information about how the application or module name is derived, see naming-standards.

Use the `--altdd` or `--runtimealtdd` options of the `deploy` (and `redeploy`) subcommand to deploy an application or module using a top-level alternate deployment descriptor. The `--altdd` option specifies a top-level alternate Jakarta EE standard deployment descriptor.

The `--runtimealtdd` option specifies a top-level alternate Payara Server runtime deployment descriptor. See "this example" for more information.

For more information about deployment descriptors associated with Payara Server, see deployment-descriptors-and-annotations.

You can also specify the deployment order of an application by using the `--deploymentorder` option of the `deploy` (and `redeploy`) subcommand. This is useful for applications that must be loaded in a certain order at server startup. Applications with lower deployment order numbers are loaded first. See "this example" for more information.

If a deployment order is not specified at the time an application is deployed, the default deployment order of `100` is assigned. If two applications have the same deployment order, the application that was deployed first is loaded first at server startup.

1. Ensure that the server is running. Remote commands require a running server.
2. List deployed applications by using the "`list-applications`" subcommand.
3. Deploy the application or module by using the "`deploy`" subcommand.

Information about the options and properties of the subcommand is included in this

help page.

4. If needed, fix issues and rerun the `deploy` subcommand.

Example - Deploying an Enterprise Application

This example deploys `newApp.ear` to the default server, `server`.

```
asadmin> deploy Cart.ear
Application deployed successfully with name Cart.
Command deploy executed successfully
```

Example - Deploying a Connector Module

This example deploys a connector module that is packaged in an RAR file.

```
asadmin> deploy jdbcra.rar
Application deployed successfully with name jdbcra.
Command deploy executed successfully
```

Example - Using an Alternate Jakarta EE Standard Deployment Descriptor

This example deploys an application using an alternate Jakarta EE standard deployment descriptor file that resides outside an application archive. Specify an absolute path or a relative path to the alternate deployment descriptor file.

```
asadmin> deploy --altdd path_to_alternate_descriptor cart.ear
Application deployed successfully with name cart.
Command deploy executed successfully
```

Example - Specifying the Deployment Order of an Application

This example specifies the deployment order of two applications. The `cart` application is loaded before the `horse` application at server startup.

Some lines of output are omitted from this example for readability.

```
asadmin deploy --deploymentorder 102 --name cart cart.war

asadmin deploy --deploymentorder 110 --name horse horse.war
```

TIP	You can also view the full syntax and options of the subcommand by typing

> `asadmin help deploy` at the command line.

To Change Targets for a Deployed Application or Module

After deployment, the deployed application or module exists in the central repository and can be referenced by the server instances or clusters that you deployed to as targets.

The `asadmin create-application-ref` and `asadmin delete-application-ref` subcommands enable you to add or delete targets for a deployed component.

Because the application or module itself is stored in the central repository, adding or deleting targets adds or deletes the same version of the component on different targets.

1. Ensure that the server is running. Remote commands require a running server.
2. Add and remove targets by using the "`create-application-ref`" and "`delete-application-ref`" subcommands.

> **TIP** You can also view the full syntax and options of the subcommand by typing`asadmin help create-application-ref` or `asadmin help delete-application-ref` at the command line.

To List Deployed Applications or Modules

There are a number of commands that can be used to list deployed applications or modules and their subcomponents. Use the commands in this section in remote mode.

1. Ensure that the server is running. Remote commands require a running server.
2. List the desired applications by using the "`list-applications`" subcommand or the "`list-sub-components`" subcommand.

   Information about these commands is included in these help pages.
3. Show the status of a deployed component by using the "`show-component-status`" subcommand.

Example Listing Applications

The `list-applications` subcommand lists all deployed Jakarta EE applications or modules. If the `--type` option is not specified, all components are listed.

This example lists deployed applications.

```
asadmin> list-applications --type web
hellojsp <web>
Command list-applications executed successfully
```

Example - Listing Subcomponents

The `list-sub-components` subcommand lists EJBs or servlets in a deployed module or in a module of the deployed application. If a module is not identified, all modules are listed. The `--appname` option functions only when the given module is standalone.

To display a specific module in an application, you must specify the module name and the `--appname` option.

This example gets the subcomponents of module `mejb.jar` within application `MEjbApp`.

```
asadmin> list-sub-components --appname MEjbApp mejb.jar
MEJBBean <StatelessSessionBean>
Command list-sub-components executed successfully
```

Example - Showing Status of a Deployed Component

The `show-component-status` subcommand gets the status (enabled or disabled) of the deployed component. This example gets the status of the `MEjbApp` component.

```
asadmin show-component-status MEjbApp
Status of MEjbApp is enabled
Command show-component-status executed successfully
```

To Redeploy an Application or Module

Use the `redeploy` subcommand in remote mode to overwrite a previously-deployed application or module. You can also accomplish this task by using the `--force` option of the `deploy` subcommand. Whenever a redeployment is done, the HTTP and SFSB

sessions in transit at that time, and the EJB timers, become invalid unless you use the `--keepstate=true` option of the `redeploy` subcommand.

Before You Begin

You must remove a preconfigured resource before it can be updated.

1. Ensure that the server is running.

   Remote commands require a running server.

2. Redeploy an application or module by using the "`redeploy`" subcommand or the "`deploy`" subcommand with the `--force` option.

   Information about the options and properties of these commands is included in these help pages.

Example - Retaining HTTP Session State During Redeployment

This example redeploys the `hello` web application. In a production environment, you usually want to retain sessions.

If you use the `--keepstate` option, active sessions of the application are retained and restored when redeployment is complete.

```
asadmin> redeploy --name hello --keepstate=true hello.war
Application deployed successfully with name hello.
Command redeploy executed successfully.
```

Keep State is a checkbox option when you redeploy using the Administration Console. For instructions, see the Administration Console online help.

Example - Redeploying a Web Application that Was Deployed from a Directory

This example redeploys the `hello` web application, which was originally deployed from the `hellodir` directory.

```
asadmin>redeploy --name hellodir
Application deployed successfully with name hellodir.
Command redeploy executed successfully.
```

Example - Redeploying an Application by Using `asadmin deploy --force`

The `--force` option is set to `false` by default. This example redeploys `newApp.ear` even if has been deployed or already exists.

```
asadmin> deploy --force=true newApp.ear
Application deployed successfully with name newApp.
Command deploy executed successfully.
```

TIP You can also view the full syntax and options of the subcommand by typing `asadmin help redeploy` at the command line.

To Disable an Application or Module

Use the `disable` subcommand in remote mode to immediately deactivate a deployed application or module without removing it from the server.

Disabling a component makes the component inaccessible to clients. However, the component is not overwritten or uninstalled, and can be enabled by using the `asadmin enable` subcommand.

An application or module is enabled by default.

1. Ensure that the server is running. Remote commands require a running server.
2. Obtain the exact name of the application or module that you are disabling.
   To list deployed applications or modules, use the "`list-applications`" subcommand. If you do not specify a type, all deployed applications and modules are listed. For example, valid types can be `web`, `ejb`, `connector`, `application`, and `webservice`.
   To see the status of deployed components, use the "`show-component-status`" subcommand.
3. Deactivate the application or module by using the "`disable`" subcommand.

   Information about the options and properties of the subcommand is included in this help page.

Example - Listing Deployed Web Applications

This example lists all deployed web applications.

```
asadmin> list-applications --type web
hellojsp <web>
Command list-applications executed successfully.
```

Example 2-12 Disabling a Web Application

This example disables the `hellojsp` application.

```
asadmin> disable hellojsp
Command disable executed successfully.
```

TIP You can also view the full syntax and options of the subcommand by typing `asadmin help disable` at the command line.

To Enable an Application or Module

An enabled application or module is runnable and can be accessed by clients if it has been deployed to an accessible server instance or cluster. An application or module is enabled by default.

Use the `enable` subcommand in remote mode to enable an application or module that has been disabled.

An application or module that is deployed to more than one target can be enabled on one target and disabled on another. If a component is referenced by a target, it is not available to users unless it is enabled on that target.

1. Ensure that the server is running. Remote commands require a running server.
2. Enable the application or module by using the "`enable`" subcommand.

   If the component has not been deployed, an error message is displayed. If the component is already enabled, it is re-enabled.

   To see the status of deployed components, use the "`show-component-status`" subcommand.

Information about the options and properties of the subcommand is included in this help page.

Example - Enabling an Application

This example enables the `sampleApp` application.

```
asadmin> enable sampleApp
Command enable executed successfully.
```

TIP You can also view the full syntax and options of the subcommand by typing `asadmin help enable` at the command line.

To Undeploy an Application or Module

Use the `undeploy` subcommand in remote mode to uninstall a deployed application or module and remove it from the repository. To reinstate the component, you must deploy the component again using the `deploy` subcommand.

1. Ensure that the server is running. Remote commands require a running server.
2. Obtain the exact name of the application or module you are un-deploying.

   To list deployed applications or modules, use the "`list-applications`" subcommand.

   If you do not specify a type, all deployed applications and modules are listed. For example, valid types can be `web`, `ejb`, `connector`, `application`, and `webservice`.

   To see the status of deployed components, use the "`show-component-status`" subcommand.
3. Undeploy the application or module by using the "`undeploy`" subcommand.

   Information about the options and properties of the subcommand is included in this help page.

Example 2-14 Listing Deployed Applications or Modules

This example lists all applications of type `web`.

```
asadmin> list-applications --type web
hellojsp <web>
Command list-applications executed successfully.
```

Example - Un-Deploying an Application

This example uninstalls the `hellojsp` application.

```
asadmin> undeploy hellojsp
hellojsp <web>
Command undeploy executed successfully.
```

TIP You can also view the full syntax and options of the subcommand by typing `asadmin help undeploy` at the command line.

To Reload Changes to Applications or Modules Dynamically

Dynamic reloading enables you to change the code or deployment descriptors of an application or module without needing to perform an explicit redeployment.

Instead, you can copy the changed class files or descriptors into the deployment directory for the application or module. The server checks for changes periodically and automatically redeploys the changes if the timestamp of the `.reload` file in the root directory for the application or module has changed.

NOTE Dynamic reloading is enabled by default, and is available only on the default server instance.

1. Go to the root directory of the deployed application or module.

   For an application:

   ```
   domain-dir/applications/app-name
   ```

   For an individually deployed module:

```
domain-dir/applications/module-name
```

NOTE | Deployment directories might change between Payara Server releases.

2. Create or update the timestamp of the `.reload` file to load the changes.

   For UNIX: `touch .reload`

   For Windows: `echo> .reload`

   If the `.reload` file doesn't exist, the `touch` or `echo` command creates it.

To Deploy an Application or Module Automatically

WARNING | This task is best suited for use in a development environment.

Automatic deployment involves copying an archive file into a special auto-deploy directory where the archive is automatically deployed by Payara Server at predefined intervals.

This method is useful in a development environment because it allows new code to be tested quickly. Automatic deployment is enabled by default, and is available only on the default server instance.

1. Use the "`set`" subcommand to adjust the auto-deployment interval.

   This sets the interval at which applications and modules are checked for code changes and dynamically reloaded. The default is `2`.

2. Use the "`set`" subcommand to enable JSP precompilation.
3. Copy your archive file to the auto-deploy directory.

   The default location is `domain-dir/autodeploy`. The application will be deployed at the next interval.

To undeploy an automatically deployed application or module, remove its archive file from the auto-deploy directory.

NOTE	Deployment directories might change between Payara Server releases.

Example - Setting the Auto-Deployment Interval

This example sets the auto-deployment interval to 3 seconds (default is 2).

```
asadmin set server.admin-service.das-config.autodeploy-polling-interval-in-seconds
=3
Command set executed successfully.
```

Example - Setting JSP Precompilation

This example enables JSP precompilation (default is `false`).

```
asadmin>
set server.admin-service.das-config.autodeploy-jsp-precompilation-enabled=true
Command set executed successfully.
```

TIP	You can also view the full syntax and options of the subcommand by typing `asadmin set --help` at the command line.

To Deploy an Application or Module by Using a Deployment Plan

In the deployment plan for an EAR file, the `payara-application.xml` file is located at the root. The deployment descriptor for each module is stored according to this syntax: `module-name.gf-dd-name`, where the `gf-dd-name` depends on the module type.

Each `/` (forward slash) is replaced by a `#` (pound sign).

1. Ensure that the server is running. Remote commands require a running server.
2. Deploy the application or module by using the "`deploy`" subcommand with the `--deploymentplan` option.

NOTE	Deployment directories might change between Payara Server releases.

Example - Deploying by Using a Deployment Plan

This example deploys the application in the `myrostapp.ear` file according to the plan specified by the `mydeployplan.jar` file.

```
asadmin deploy --deploymentplan mydeployplan.jar myrostapp.ear

Application deployed successfully with name myrostapp.
Command deploy executed successfully.
```

Example - Deployment Plan Structure for an Enterprise Application

This listing shows the structure of the deployment plan JAR file for an EAR file.

```
$ jar -tvf mydeployplan.jar
420 Thu Mar 13 15:37:48 PST 2003 glassfish-application.xml
370 Thu Mar 13 15:37:48 PST 2003 RosterClient.war.payara-web.xml
418 Thu Mar 13 15:37:48 PST 2003 roster-ac.jar.glassfish-application-client.xml
1281 Thu Mar 13 15:37:48 PST 2003 roster-ejb.jar.glassfish-ejb-jar.xml
2317 Thu Mar 13 15:37:48 PST 2003 team-ejb.jar.glassfish-ejb-jar.xml
84805 Thu Mar 13 15:37:48 PST 2003 team-ejb.jar.RosterSchema.dbschema
```

To Deploy an Application or Module in a Directory Format

WARNING	This task is best suited for use in a development environment.

An expanded directory, also known as an exploded directory, contains an un-assembled (un-packaged) application or module. To deploy a directory format instead of an archive, file, use the `asadmin deploy` subcommand in remote mode and specify a path to a directory instead of to an archive file.

The contents of the directory must be the same as the contents of a corresponding archive file, with one exception.

An application archive file contains archive files for its modules, for example `myUI.war` and `myEJB.jar`.

The expanded application directory contains expanded directories for the modules, for

example `myUI_war` and `myEJB_jar`, instead.

You can change deployment descriptor files directly in the expanded directory.

If your environment is configured to use dynamic reloading, you can also dynamically reload applications or modules that are deployed from the directory. For instructions, see "To Reload Changes to Applications or Modules Dynamically".

Unlike archive file deployment, directory deployment does not copy the directory contents to the remote hosts.

This means that for deployment to a cluster, the directory path may exist for both the DAS and the remote server instances but may not actually correspond to the same physical location. If any target server instance cannot see the deployed directory, or finds that it contains different files from those detected by the DAS, deployment fails.

Integrated development environments (IDEs) typically use directory deployment, so you do not need to deal directly with the expanded format.

Before You Begin

On each cluster or stand-alone server instance to which the application or module is deployed, the directory must be accessible and must contain the same files as found by the DAS.

On Windows, if you are deploying a directory on a mapped drive, you must be running Payara Server as the same user to which the mapped drive is assigned. This enables Payara Server to access the directory.

1. Ensure that the server is running. Remote commands require a running server.
2. Verify that the expanded directory contents match the archive file.

   For information about the required directory contents, see the appropriate specifications.

3. Deploy the directory by using the "`deploy`" subcommand and specifying the path to the expanded directory.

NOTE	Deployment directories might change between Payara Server releases.

Example - Deploying an Application from a Directory

This example deploys the expanded directory `/apps/MyApp` for the `hello` application.

```
asadmin> deploy --name hello /apps/MyApp
Application deployed successfully with name hello.
Command deploy executed successfully.
```

TIP	You can also view the full syntax and options of the subcommand by typing `asadmin help deploy` at the command line.

To Deploy an Application on Server Start

As described in To Deploy an Application or Module Automatically Payara Server has the ability to automatically deploy applications on server start using the `autodeploy` directory. While acceptable in development environments, the internal scanner used to deploy files in this directory is a security risk for production environments.

To introduce the capability of deploying an application on server startup without the risk of enabling the deployment scanner, Payara Server has the ability to run asadmin commands at startup via command scripts, either before or after the server's boot sequence has been completed.

To enable this feature, the `start-domain` Asadmin command accepts two extra options: `--prebootcommandfile` and `--postbootcommandfile` which can be used as follows:

Pre-boot command script usage

```
asadmin start-domain --prebootCommandFile=/path/to/file
```

IMPORTANT	Due to the time during boot when the commands in this file are executed, this only supports `set` commands.

Post-boot command script usage

```
asadmin start-domain --postbootCommandFile=/path/to/file
```

To deploy an application at startup time, simply create a command script that deploys the application like so:

deployment-script.asadmin

```
deploy --name MyApp /path/to/deployment/archive
```

And use this script file with the `--postbootCommandFile` option:

Post-boot command script usage

```
asadmin start-domain --postbootCommandFile=deployment-script.asadmin
```

IMPORTANT	Post boot command scripts will guarantee that applications will be deployed AFTER the server has completed its boot sequence, guaranteeing that all required resources will be available for the application to use.

Command Script Considerations

There are some considerations to have on how to structure command scripts:

- Script files should be saved in a text format, with each command on its own line.
- Lines where the first non-whitespace character is `#` are treated as comments
- Any blank lines will be ignored
- Any invalid or failing commands will be logged to the server log and the file will continue to be processed.
- Script files ***must*** exist, and be accessible by the server instance.

- Both pre-boot and post-boot script files can be passed at the same time when starting the domain.

To Deploy from a URL

Payara Server also has the ability to deploy applications whose archives are hosted on a remote location. To deploy an application archive that can be reached via its URL, use the "`deploy-remote-archive`" subcommand, which works similarly to the standard `deploy` command, and can be used in the same way.

The only difference in the command is that it takes a valid URL instead of a file location in the host's filesystem.

Example - Deploying by Using a URL

The following example showcases how to deploy an application's archive for the `clusterjsp` application hosted in the `https://nexus.payara.fish/repository/payara-artifacts` remote repository using the `deploy-remote-archive` subcommand:

```
asadmin deploy-remote-archive https://nexus.payara.fish/repository/payara-
artifacts/fish/payara/testing/clusterjsp/1.1/clusterjsp-1.1.war
```

WARNING | The command will only accept HTTP and HTTPS URLs.

To Deploy Using a Maven Artifact

Payara Server also has the ability to deploy applications whose archives are hosted as Maven artifacts on remote repositories. To deploy an application archive in this manner, use the "`deploy-remote-archive`" subcommand, which works similarly to the standard `deploy` command, and point it to a set of Maven GAV (Group ID/Artifact ID/Version) coordinates where the artifact is located.

By default, the command will search for the artifact in Maven Central, but specific Maven repositories can be added as search locations by using the `additionalRepositories` option, which accepts a comma separated list of repository URLs.

Example - Deploying by Using a Maven Artifact

The following example showcases how to deploy an application's archive for the `clusterjsp-1.1.1.war` application artifact which is located in the `https://nexus.payara.fish/repository/payara-artifacts` Maven repository using the `deploy-remote-archive` subcommand:

NOTE

If an application matching the GAV coordinates exists on multiple repositories, the first found will be deployed. The order that the repositories are searched is the order that you specify them in.
For example, given an `additionalRepositories` of *repo1,repo2*, the order that the repositories would be searched is: Maven Central repo1 repo2.

```
asadmin deploy-remote-archive --additionalRepositories
https://nexus.payara.fish/repository/payara-artifacts
fish.payara.testing,clusterjsp,1.1
```

Modifying the Configuration of a Web Application or Module

You can modify the configuration of a web application or a module by modifying the deployment descriptors and then repackaging and redeploying the application.

The instructions in this section enable you to change the configuration of a deployed application without the need to modify the application's deployment descriptors and repackage and redeploy the application. If the application or module entry is unset, its value reverts to the value, if any, that is set in the application's deployment descriptor.

The following topics are addressed here:

- "To Set a Web Context Parameter"
- "To Unset a Web Context Parameter"
- "To List Web Context Parameters"
- "To Set a Web Environment Entry"
- "To Unset a Web Environment Entry"

- "To List Web Environment Entries"

To Set a Web Context Parameter

Use the `set-web-context-param` subcommand in remote mode to change the configuration of a deployed application without the need to modify the application's deployment descriptors and repackage and redeploy the application. By using this subcommand, you are either adding a new parameter that did not appear in the original web module's descriptor, or overriding the descriptor's setting of the parameter.

If the `--ignoreDescriptorItem` option is set to `true`, then the server ignores any setting for that context parameter in the descriptor, which means you do not need to specify an overriding value on the `set-web-context-param` subcommand.

The server behaves as if the descriptor had never contained a setting for that context parameter.

This subcommand sets a servlet context-initialization parameter of one of the following items:

- A deployed web application
- A web module in a deployed Jakarta EE application

Before You Begin

The application must already be deployed. Otherwise, an error occurs.

1. Ensure that the server is running.

   Remote commands require a running server.

2. Set a servlet context-initialization parameter by using the "`set-web-context-param`" subcommand.

   Information about the options for the subcommand is included in this help page.

Example - Setting a Servlet Context-Initialization Parameter for a Web Application

This example sets the servlet context-initialization parameter `jakarta.faces.STATE_SAVING_METHOD` of the web application `basic-ezcomp` to client.

```
asadmin> set-web-context-param --name= jakarta.faces.STATE_SAVING_METHOD
--description="The location where the application?s state is preserved" --value
=client basic-ezcomp

Command set-web-context-param executed successfully.
```

TIP You can also view the full syntax and options of the subcommand by typing `asadmin help set-web-context-param` at the command line.

To Unset a Web Context Parameter

Use the `unset-web-context-param` subcommand in remote mode to unset an environment entry for a deployed web application or module that has been set by using the `set-web-env-entry` subcommand. There is no need to modify the application's deployment descriptors and repackage and redeploy the application.

This subcommand unsets an environment entry for one of the following items:

- A deployed web application
- A web module in a deployed Jakarta EE application

When an entry is unset, its value reverts to the value, if any, that is set in the application's deployment descriptor. This subcommand cannot be used to change the value of an environment entry that is set in an application's deployment descriptor.

Instead, use the "`set-web-context-param`" subcommand for this purpose.

Before You Begin

The application must already be deployed, and the entry must have previously been set by using the `set-web-env-entry` subcommand. Otherwise, an error occurs.

1. Ensure that the server is running.

   Remote commands require a running server.

2. Unset an environment entry by using the "`unset-web-context-param`" subcommand.

   Information about the options for the subcommand is included in this help page.

Example - Unsetting a Servlet Context-Initialization Parameter for a Web Application

This example unsets the servlet context-initialization parameter ` jakarta.faces.STATE_SAVING_METHOD` of the web application `basic-ezcomp`.

```
asadmin unset-web-context-param --name= jakarta.faces.STATE_SAVING_METHOD basic-
ezcomp
Command unset-web-context-param executed successfully.
```

TIP You can also view the full syntax and options of the subcommand by typing `asadmin help unset-web-context-param` at the command line.

To List Web Context Parameters

Use the `list-web-context-param` subcommand in remote mode to list the parameters that have previously been set by using the "`set-web-context-param`" subcommand.

The subcommand does not list parameters that are set only in the application's deployment descriptor. For each parameter, the following information is displayed:

- The name of the parameter
- The value to which the parameter is set
- The value of the `--ignoreDescriptorItem` option of the `set-web-context-param` subcommand that was specified when the parameter was set
- The description of the parameter or `null` if no description was specified when the parameter was set

1. Ensure that the server is running.

   Remote commands require a running server.

2. List servlet context-initialization parameters by using the "`list-web-context-param`" subcommand.

Example - Listing Servlet Context-Initialization Parameters for a Web Application

This example lists all servlet context-initialization parameters of the web application `basic-ezcomp` that have been set by using the `set-web-context-param` subcommand. Because no description was specified when the `jakarta.faces.PROJECT_STAGE` parameter was set, null is displayed instead of a description for this parameter.

```
asadmin list-web-context-param basic-ezcomp jakarta.faces.STATE_SAVING_METHOD =
client ignoreDescriptorItem=false

jakarta.faces.PROJECT_STAGE = null ignoreDescriptorItem=true
Command list-web-context-param executed successfully.
```

TIP You can also view the full syntax and options of the subcommand by typing `asadmin help list-web-context-param` at the command line.

To Set a Web Environment Entry

An application uses the values of environment entries to customize its behavior or presentation. Use the `set-web-env-entry` subcommand in remote mode to change the configuration of a deployed application without the need to modify the application's deployment descriptors and repackage and redeploy the application.

By using this subcommand, you are either adding a new parameter that did not appear in the original web module's descriptor, or overriding the descriptor's setting of the parameter.

If you the `--ignoreDescriptorItem` option is set to `true`, then the server ignores any setting for that environment entry in the descriptor, which means you do not need to specify an overriding value on the `set-web-env-entry` subcommand. The server

behaves as if the descriptor had never contained a setting for that environment entry.

This subcommand sets an environment entry for one of the following items:

- A deployed web application
- A web module in a deployed Jakarta EE application

Before You Begin

The application must already be deployed. Otherwise, an error occurs.

1. Ensure that the server is running.

   Remote commands require a running server.

2. Set an environment entry for a deployed web application or module by using the "`set-web-env-entry`" subcommand.

   Information about the options for the subcommand is included in this help page.

Example - Setting an Environment Entry for a Web Application

This example sets the environment entry `Hello User` of the application hello to `techscribe`. The Java type of this entry is `java.lang.String`.

```
asadmin> set-web-env-entry --name="Hello User" --type=java.lang.String --value
=techscribe --description="User authentication for Hello appplication" hello

Command set-web-env-entry executed successfully
```

TIP You can also view the full syntax and options of the subcommand by typing `asadmin help set-web-env-entry` at the command line.

To Unset a Web Environment Entry

Use the `unset-web-env-entry` subcommand in remote mode to unset an environment entry for a deployed web application or module.

1. Ensure that the server is running.

Remote commands require a running server.

2. Unset a web environment entry by using the "`unset-web-env-entry`" subcommand.

   Information about the options for the subcommand is included in this help page.

Example - Unsetting an Environment Entry for a Web Application

This example unsets the environment entry `Hello User` of the web application `hello`.

```
asadmin unset-web-env-entry --name="Hello User" hello
Command unset-web-env-entry executed successfully.
```

To List Web Environment Entries

Use the `list-web-env-entry` subcommand to list environment entries for a deployed web application or module. For each entry, the following information is displayed:

- The name of the entry
- The Java type of the entry
- The value to which the entry is set
- The description of the entry or null if no description was specified when the entry was set
- The value of the `--ignoreDescriptorItem` option of the `set-web-env-entry` subcommand that was specified when the entry was set

1. Ensure that the server is running.

   Remote commands require a running server.

2. List the environment entries by using the "`list-web-env-entry`" subcommand.

Example - Listing Environment Entries for a Web Application

This example lists all environment entries that have been set for the web application `hello` by using the `set-web-env-entry` subcommand.

```
asadmin list-web-env-entry hello

Hello User (java.lang.String) = techscribe ignoreDescriptorItem=false
Hello Port (java.lang.Integer) = null ignoreDescriptorItem=true
Command list-web-env-entry executed successfully.
```

TIP	You can also view the full syntax and options of the subcommand by typing asadmin help `list-web-env-entry` at the command line.

Web Module Deployment Guidelines

The following guidelines apply to deploying a web module in Payara Server:

- **Context Root**. When you deploy a web module, if you do not specify a context root, the default is the name of the WAR file without the `.war` extension. The web module context root must be unique within the server instance.

  The domain administration server (DAS) in Payara Server supports the deployment of multiple web applications using the same web context root as long as those applications are deployed to different Payara Server stand-alone instances.

IMPORTANT	Deploying multiple applications using the same context root within a single instance produces an error.

- **Data Source**. If a web application accesses a `DataSource` that is not specified in a `resource-ref` in `glassfish-web.xml` or `payara-web.xml`, or there is no `glassfish-web.xml` or `payara-web.xml` files, the `resource-ref-name` defined in `web.xml` is used.

  A warning message is logged, recording the JNDI name that was used to look up the resource.

- **Virtual Servers**. If you deploy a web application and do not specify any assigned virtual servers, the web application is assigned to all currently-defined virtual servers except the virtual server with ID `__asadmin`, which is reserved for administrative purposes.

If you then create additional virtual servers and want to assign existing web applications to them, you must redeploy the web applications.

- **HTTP Sessions**. If a web application is un-deployed, all its HTTP sessions will be invalidated and removed, unless the application is being un-deployed as part of a redeployment and the `--keepstate` deployment option was set to true.

  This option is not supported and ignored in a clustered environment. See this example.

  For information about HTTP session persistence, see the "Payara Server High Availability section".

- **JSP Precompilation**. You can precompile JSP files during deployment by checking the appropriate box in the Administration Console, or by using the `--precompilejsp` option of the `deploy` subcommand.

  You can keep the generated source for JSP files by adding the `keepgenerated` flag to the `jsp-config` element in `glassfish-web.xml` or `payara-web.xml`.

  For example:

```
<payara-web-app>
   <jsp-config>
      <property name="keepgenerated" value="true" />
   </jsp-config>
</payara-web-app>
```

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

+ If you include this property when you deploy the WAR file, the generated source is kept in `domain-dir/generated/jsp/app-name/module-name` for an application, or `domain-dir/generated/jsp/module-name` for an individually-deployed web module.

+ For more information about JSP precompilation, see jsp-config.

- **Web Context Parameters**. You can set web context parameters after deployment. See the following sections:
  - "To Set a Web Context Parameter"
  - "To Unset a Web Context Parameter"
  - "To List Web Context Parameters"
- **Web Environment Entries**. You can set web environment entries after deployment. See the following sections:
  - "To Set a Web Environment Entry"
  - "To Unset a Web Environment Entry"
  - "To List Web Environment Entries"

EJB Module Deployment Guidelines

NOTE

The Payara Server Web Profile supports the Jakarta Enterprise Beans 4.0 Lite specification, which allows enterprise beans within web applications, among other features.
The Payara Server Full Platform Profile supports the entire Jakarta Enterprise Beans 4.0 specification. For details, see Jakarta Enterprise Beans

The following guidelines apply to deploying an EJB module in Payara Server:

- **JNDI Name**. — If no JNDI name for the EJB JAR module is specified in the `jndi-name` element immediately under the `ejb` element in `payara-ejb-jar.xml`, or there is no `glassfish-ejb-jar.xml` file, a default, non-clashing JNDI name is derived.

  A warning message is logged, recording the JNDI name used to look up the EJB JAR module.

  Because the Jakarta Enterprise Beans 4.0 specification defines portable EJB JNDI names, there is less need for Payara Server specific JNDI names. By default, Payara

Server specific default JNDI names are applied automatically for backward compatibility.

To disable Payara Server specific JNDI names for an EJB module, set the value of the `<disable-nonportable-jndi-names>` element in the `glassfish-ejb-jar.xml` file to `true`. The default is `false`.

- **Stateful Session Bean and Timer State**. — Use the `--keepstate` option of the "`redeploy`" subcommand or the `<keepstate>` element in the `glassfish-ejb-jar.xml` file to retain stateful session bean instances and persistently created EJB timers across re-deployments.

  The `--keepstate` option of the `redeploy` subcommand takes precedence. The default for both is `false`. This option is not supported and ignored in a clustered environment.

  Some changes to an application between re-deployments can prevent this feature from working properly. For example, do not change the set of instance variables in the SFSB bean class. Other examples would be changes to EJB names, or adding or removing EJBs to or from an application.

- **EJB Singletons**. — EJB Singletons are created for each server instance in a cluster, and not once per cluster or deployment group. Check the "Clustered Singleton" proprietary feature of the Public API on how to achieve this.
- **Stubs and Ties**. — Use the "`get-client-stubs`" subcommand in remote mode to retrieve stubs and ties.
- **Compatibility of JAR Visibility Requirements**. — Use the `compatibility` element of the `payara-application.xml` or `payara-ejb-jar.xml` file to specify the Payara Server release with which to be backward compatible in terms of JAR visibility requirements for applications.

  The current allowed value is `v2`, which refers to older runtimes from where Payara Server is based from.

NOTE	Starting in Java EE 6, the Java EE specification imposes stricter requirements than Java EE 5 did on which JAR files can be visible to various modules within an EAR file. Setting this element to `v2` removes these Java EE 6 and later restrictions.

Deploying a Connector Module

Deploying a stand-alone connector module allows multiple deployed Jakarta EE applications to share the connector module. A resource adapter configuration is automatically created for the connector module.

The following topics are addressed here:

- "To Deploy and Configure a Stand-Alone Connector Module"
- "Redeploying a Stand-Alone Connector Module"
- "Deploying and Configuring an Embedded Resource Adapter"

To Deploy and Configure a Stand-Alone Connector Module

As an alternative to Step 3 through Step 6, you can define application-scoped resources in the `glassfish-resources.xml` or `payara-resources.xml` deployment descriptor.

IMPORTANT	glassfish-resources.xml is deprecated and support will be removed in a future release. It is recommended to use payara-resources.xml instead.

For more information, see application-scoped-resources.

1. Ensure that the server is running.

   Remote commands require a running server.

2. Deploy the connector module by using the deploy subcommand.
3. Configure connector connection pools for the deployed connector module.

Use the `create-connector-connection-pool` subcommand. For procedures, see "to-create-a-connector-connection-pool" in the *General Administration section*.

4. Configure connector resources for the connector connection pools.

   Use the `create-resource-adapter-config` subcommand. For procedures, see "to-create-configuration-information-for-a-resource-adapter" in the *General Administration section*.

   If needed, you can override the default configuration properties of a resource adapter.

   This step associates a connector resource with a JNDI name.

5. Configure a resource adapter.

   Use the `create-resource-adapter-config` subcommand. For procedures, see "to-create-configuration-information-for-a-resource-adapter" in the *General Administration section*.

   If needed, you can override the default configuration properties of a resource adapter.

6. If needed, create an administered object for an inbound resource adapter.

   Use the `create-admin-object` subcommand. For procedures, see "to-create-an-administered-object" in the *General Administration section*.

Redeploying a Stand-Alone Connector Module

Redeployment of a connector module maintains all connector connection pools, connector resources, and administered objects defined for the previously deployed connector module. You do not need to reconfigure any of these resources.

However, you should redeploy any dependent modules. A dependent module uses or refers to a connector resource of the redeployed connector module. Redeployment of a connector module results in the shared class loader reloading the new classes. Other modules that refer to the old resource adapter classes must be redeployed to gain access to the new classes.

For more information about class loaders, see "class-loaders" in the Application Development section.

During connector module redeployment, the server log provides a warning indicating that all dependent applications should be redeployed.

Client applications or application components using the connector module's resources may throw class cast exceptions if dependent applications are not redeployed after connector module redeployment.

To disable automatic redeployment, set the `--force` option to `false`. In this case, if the connector module has already been deployed, Payara Server provides an error message.

Deploying and Configuring an Embedded Resource Adapter

A connector module can be deployed as a Jakarta EE component in a Jakarta EE application. Such connectors are only visible to components residing in the same Jakarta EE application. Deploy this application as you would any other Jakarta EE application.

You can create new connector connection pools and connector resources for a connector module embedded within a Jakarta EE application by prefixing the connector name with `app-name#`.

For example, if an application `appX.ear` has `jdbcra.rar` embedded within it, the connector connection pools and connector resources refer to the connector module as `appX#jdbcra`.

An embedded connector module cannot be un-deployed using the name `app-name#connector-name`. To undeploy the connector module, you must undeploy the application in which it is embedded.

The association between the physical JNDI name for the connector module in Payara Server and the logical JNDI name used in the application component is specified in the Payara Server-specific XML descriptor `glassfish-ejb-jar.xml`.

Assembling and Deploying an Application Client Module

Deployment is necessary for application clients that communicate with EJB components.

NOTE	The Application Client Container is supported only in the Payara Server Full Platform Profile, not in the Web Profile.

The following topics are addressed here:

- "To Assemble and Deploy an Application Client"
- "To Prepare Another Machine for Running an Application Client"
- "To Undeploy an Application Client"

To Assemble and Deploy an Application Client

1. Assemble the necessary client components.

   The client JAR file is created.

2. Assemble the EJB components that are to be accessed by the client.

   The EJB JAR file is created.

3. Assemble the client and EJB JAR files together in an EAR.

   An EAR file contains all the components of the application.

4. Deploy the application.

   Instructions are contained in to-deploy-an-application-or-module.

5. If you are using the `appclient` script to run the application client, retrieve the client files.

   The client artifacts contain the ties and necessary classes for the application client. Client artifacts include multiple files.

   You can use either the `get-client-stubs` subcommand or the `--retrieve`

option of the `deploy` subcommand, but you do not need to use both.

- Use the "`deploy`" subcommand with the `--retrieve` option to retrieve the client files as part of deploying the application.
- Use the "`get-client-stubs`" subcommand to retrieve client files for a previously-deployed application.

6. Test the client on the Payara Server machine by using the `appclient` script.

   The `appclient` script is located in the `as-install/bin` directory.

   If you are using the default server instance, the only required option is `-client`, which points to the client JAR file. For example:

   ```
   appclient -client converterClient.jar
   ```

   The `-xml` parameter, which specifies the location of the `sun-acc.xml` file, is also required if you are not using the default instance.

   TIP For more detailed information about the `appclient` script, see "`appclient`".

For more detailed information about creating application clients, see ""Developing Java Clients"" in the Application Development section. This chapter includes information on the following topics:

- Accessing EJB components and JMS resources from application clients
- Connecting to a remote EJB module through a firewall
- Using Java Web Start and creating a custom JNLP file
- Using libraries with application clients
- Specifying a splash screen, login retries, and other customizations

To Prepare Another Machine for Running an Application Client

If the *context-root* or *module-id* is not specified during deployment, the name of the EAR

or JAR file without the `.ear` or `.jar` extension is used. For an application, the relative path to the application client JAR file is also included. If the application or module is not in EAR or JAR file format, a *context-root* or *module-id* is generated.

For details about naming, see naming-standards.

Before You Begin

This task applies if you want to use the `appclient` script to run the application client on a system other than where the server runs.

1. Create the application client package JAR file.

   Use the `package-appclient` script in the `as-install/bin` directory. This JAR file is created in the `as-install/lib/appclient` directory.

2. Copy the application client package JAR file to the client machine.
3. Extract the contents of the JAR file.

   For example: `jar xf filename.jar`

4. Configure the `sun-acc.xml` file.

   If you used the `package-appclient` script, this file is located in the `appclient/appserv/lib/appclient` directory by default.

5. Configure the `asenv.conf` (`asenv.bat` on Windows) file.

   This file is located in `appclient/appserv/bin` by default if you used the `package-appclient` script.

6. Copy the client JAR file to the client machine.

   You are now ready to run the client.

TIP For more detailed information about the `package-appclient` script, see "`appclient`(1M)".

To Undeploy an Application Client

After application clients are downloaded, they remain on the client until they are manually removed.

If you undeploy an application client, you can no longer download that application client because it might be in an inconsistent state. If you try to launch an application client that was previously downloaded (even though the server side of the application client is no longer present), the results are unpredictable unless the application client has been written to tolerate such situations.

Web Service Deployment Guidelines

NOTE	If you installed the Web Profile, web services are not supported unless the optional Metro Web Services Stack add-on component is downloaded. Without the Metro add-on component, a servlet or EJB component cannot be a web service endpoint, and the `glassfish-web.xml` /`payara-web.xml` and `glassfish-ejb-jar.xml` elements related to web services are ignored.

The following guidelines apply when deploying a web service in Payara Server:

- **Web Service Endpoint**. Deploy a web service endpoint to Payara Server as you would any servlet or stateless session bean.

  If the deployed application or module has a web service endpoint, the endpoint is detected automatically during deployment.

  The Payara Server specific deployment descriptor files, `glassfish-web.xml`, `payara-web.xml` and `glassfish-ejb-jar.xml`, provide optional web service enhancements in their webservice-endpoint and webservice-description elements.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

- **Web Service Management**. Web service management is fully supported in the Administration Console. After the application or module is deployed, click the Web Service component. The table in the right frame lists deployed web service endpoints.

For more information about web services, see "developing-web-services" in the Application Development section.

Osgi Bundle Deployment Guidelines

To deploy an OSGi bundle using the Administration Console, select Other from the Type drop-down list and check the OSGI Type checkbox.

To deploy an OSGi bundle using the `asadmin deploy` command, set the `--type` option to the value `osgi`. For example:

```
asadmin deploy --type=osgi MyBundle.jar
```

To automatically deploy an OSGi bundle, copy the bundle archive to the `domain-dir/autodeploy/bundles` directory.

NOTE

For components packaged as OSGi bundles (`--type=osgi`), the `deploy` subcommand accepts properties arguments to wrap a WAR file as a WAB (Web Application Bundle) at the time of deployment. The subcommand looks for a key named `UriScheme` and, if present, uses the key as a URL stream handler to decorate the input stream.

Other properties are used in the decoration process. For example, the Payara Server OSGi web container registers a URL stream handler named `webbundle`, which is used to wrap a plain WAR file as a WAB.

Transparent JDBC Connection Pool Reconfiguration

In this Payara Server release, reconfiguration of a JDBC connection pool due to attribute or property changes can be transparent to the applications or modules that use the pool, even if pool reconfiguration results in pool recreation. You do not need to redeploy

the application or module.

To enable transparent pool reconfiguration, set the `dynamic-reconfiguration-wait-timeout-in-seconds` property. This property specifies the timeout for dynamic reconfiguration of the pool.

In-progress connection requests must complete before this timeout expires, or they must be retried. New connection requests wait for this timeout to expire before acquiring connections to the reconfigured pool. If this property exists and has a positive value, it is enabled.

You can set this property in the `glassfish-resources.xml` or `payara-resources.xml` file. For more information, see the property descriptions under jdbc-connection-pool.

IMPORTANT	glassfish-resources.xml is deprecated and support will be removed in a future release. It is recommended to use payara-resources.xml instead.

For JDBC connection pools that are not application-scoped, use the `set` subcommand to set this property. For example, to configure `mypool` on `myserver`, type the following all on one line:

```
asadmin set myserver.resources.jdbc-connection-pool.mypool.property.dynamic-reconfiguration-wait-timeout-in-seconds=30
```

Application-Scoped Resources

You can define an application-scoped JDBC resource or other resource for an enterprise application, web module, EJB module, connector module, or application client module. This allows single-step deployment for resource-dependent modules and applications.

An application-scoped resource has the following characteristics:

- It is available only to the module or application that defines it.

- It cannot be referenced or looked up by other modules or applications.
- It is created during deployment, destroyed during un-deployment, and recreated during redeployment.
- It is free from unexpected resource starvation or delay in acquiring connections because no other application or module competes for accesses to it.

The following resource types can be application-scoped:

- JDBC connection pools
- JDBC resources
- Connector connection pools
- Connector resources
- Resource adapters
- External JNDI resources
- Custom resources
- Admin object resources
- Jakarta resources

Deployment Descriptor: An application-scoped resource is defined in the `glassfish-resources.xml` or `payara-resources.xml` deployment descriptor file. This file is placed in the `META-INF` directory of the module or application archive.

For web applications or modules, this file is placed in the `WEB-INF` directory. If any submodule archives of an enterprise application archive have their own `glassfish-resources.xml` or `payara-resources.xml` files, the resource definitions are scoped to those modules only.

For more information about the `glassfish-resources.xml` and `payara-resources.xml` file, see the-payara-resources.xml-file and "Elements of the Payara Server Deployment Descriptors".

IMPORTANT	glassfish-resources.xml is deprecated and support will be removed in a future release. It is recommended to use payara-resources.xml instead.

Naming: Application-scoped resource JNDI names begin with `java:app` or `java:module`. If one of these prefixes is not specified in the JNDI name, it is added. For example, application-scoped databases have JNDI names in the following format: `java:app/jdbc/DataSourceName` or `java:module/jdbc/DataSourceName`.

Errors: Application-scoped resource definitions with same resource name, resource type, attributes, and properties are duplicates. These generate `WARNING` level log messages and deployment continues.

Definitions with the same resource name and type but different attributes or properties are conflicts and cause deployment failure. When an application or module tries to look up a scoped resource that does not belong to it, a naming exception is thrown.

Redeployment: When an application or module is un-deployed, its scoped resources are deleted. During redeployment, resources are destroyed and recreated based on changes in the `glassfish-resources.xml` or `payara-resources.xml` file.

To preserve old resource definitions during redeployment, use the `preserveAppScopedResources` property of the `redeploy` (or `deploy --force=true`) subcommand. For example:

```
asadmin> redeploy --property preserveAppScopedResources=true MyApp.ear

asadmin> deploy --force=true --property preserveAppScopedResources=true MyApp.ear
```

For more information, see "`redeploy`" and "`deploy`".

Listing: Use the `--resources` option of the `list-applications` subcommand to list application-scoped resources. Use the `--subcomponents` option in addition to list scoped resources for enterprise application modules or for module subcomponents.

To list scoped resources for subcomponents only, use the `--resources` option of the `list-subcomponents` subcommand

For more information, see "`list-applications`" and "`list-sub-components`".

Restrictions. Use of application-scoped resources is subject to the following restrictions:

- `resource-adapter-config` and `connector-work-security-map` — These can only be specified in the `glassfish-resources.xml` or `payara-resources.xml` file of the corresponding connector module.

  In an enterprise application, the `resource-adapter-config` or `connector-work-security-map` for an embedded connector module must be specified in the `glassfish-resources.xml` or `payara-resources.xml` file of the connector module.

  You cannot specify a `resource-adapter-config` or `connector-work-security-map` in an application for a connector module that is not part of the application.

- Resource to connection pool cross references — A module-level `jdbc-resource` cannot reference an application-level `jdbc-connection-pool`. Likewise, a module-level `connector-resource` cannot reference an application-level `connector-connection-pool`.
- Global resources — Defining `java:global` JNDI names is not supported.
- Cross definitions — Defining `java:app` JNDI names at the module level is not supported.

The `asadmin` Deployment Subcommands

This appendix lists the `asadmin` deployment subcommands that are included with this release of the Payara Server software. For information on additional `asadmin` subcommands, see "subcommands-for-the-asadmin-utility" in the *Payara Server General Administration section*.

"`add-library`"

Adds one or more library JAR files to Payara Server. You can specify whether the libraries are added to the Common class loader directory, the Jakarta optional package directory, or the application-specific class loader directory.

"`create-application-ref`"

Creates a reference from a cluster or an un-clustered server instance to a previously deployed Jakarta EE application or module. This effectively results in the application element being deployed and made available on the targeted instance or cluster.

"`delete-application-ref`"

Removes a reference from a cluster or an un-clustered server instance to a previously deployed Jakarta EE application or module. This effectively results in the application element being un-deployed on the targeted instance or cluster.

"`deploy`"

Deploys an enterprise application, web application, EJB module, connector module, or application client module. If the component is already deployed or already exists, you can forcefully redeploy if you set the `--force` option to `true`.
A directory can also be deployed. Supported in remote mode only. For usage instructions, see to-deploy-an-application-or-module.

"`deploy-remote-archive`"

Deploys an enterprise application, web application, EJB module, connector module, or application client module from a remote location, either using a URL or a set of Maven GAV coordinates.
For usage instructions, see to-deploy-from-a-url and to-deploy-using-maven-artifact.

"`disable`"

Immediately deactivates the named application or module. If the component has not been deployed, an error message is returned. Supported in remote mode only.
For usage instructions, see to-disable-an-application-or-module.

"`enable`"

Enables the specified application or module. If the component has not been deployed, an error message is returned. If the component is already enabled, then it is re-enabled. Supported in remote mode only. For usage instructions, see to-enable-an-application-or-module.

"`get-client-stubs`"
Gets the client stubs JAR file for an application client module or an application containing the application client module, from the server machine to the local directory. For usage instructions, see ejb-module-deployment-guidelines.

"`list-applications`"
Lists deployed Jakarta EE applications and modules. Optionally lists subcomponents and scoped resources. If the `--type` option is not specified, all applications and modules are listed. Supported in remote mode only.
For usage instructions, see to-list-deployed-applications-or-modules.

"`list-application-refs`"
Lists Jakarta EE applications and modules deployed on the specified target server instance or cluster.

"`list-libraries`"
Lists library JAR files that have been added to Payara Server. You can specify whether to list libraries in the Common class loader directory, the Jakarta optional package directory, or the application-specific class loader directory.

"`list-sub-components`"
Lists EJBs or servlets in a deployed module or in a module of the deployed application. If a module is not identified, all modules are listed. To display a specific module in an application, you must specify the module name and the `--appname` option. Supported in remote mode only. For usage instructions, see to-list-deployed-applications-or-modules.

"`list-web-context-param`"
Lists servlet context-initialization parameters of a deployed web application or module.

Supported in remote mode only.

For usage instructions, see to-list-web-context-parameters.

"`list-web-env-entry`"

Lists environment entries for a deployed web application or module. Supported in remote mode only.

For usage instructions, see to-list-web-environment-entries.

"`redeploy`"

Overwrites an application or module that is already deployed. Supported in remote mode only.

For usage instructions, see to-redeploy-an-application-or-module.

"`remove-library`"

Removes one or more library JAR files from Payara Server. You can specify whether the libraries are removed from the Common class loader directory, the Jakarta optional package directory, or the application-specific class loader directory.

"`set-web-context-param`"

Sets a servlet context-initialization parameter of a deployed web application or module. Supported in remote mode only.

For usage instructions, see to-set-a-web-context-parameter.

"`set-web-env-entry`"

Sets an environment entry for a deployed web application or module.

Supported in remote mode only. For usage instructions, see "To Set a Web Environment Entry".

"`show-component-status`"

Shows the status of a deployed component. The possible statuses include `enabled` or `disabled`. Supported in remote mode only.

For usage instructions, see to-set-a-web-environment-entry.

"`undeploy`"

Uninstalls the specified deployed application or module. Supported in remote mode only.
For usage instructions, see to-undeploy-an-application-or-module.

"`unset-web-context-param`"
Unsets a servlet context-initialization parameter of a deployed web application or module. Supported in remote mode only.
For usage instructions, see to-unset-a-web-context-parameter.

"`unset-web-env-entry`"
Unsets an environment entry for a deployed web application or module. Supported in remote mode only.
For usage instructions, see to-unset-a-web-environment-entry.

Azul Payara Deployment Descriptor Files

This appendix describes the element hierarchies in the Azul Payara deployment descriptors that are supported on top of the standard Jakarta EE deployment descriptors.

About the Azul Payara Deployment Descriptors

Each deployment descriptor XML file has a corresponding Document Type Definition (DTD) file, which defines the elements, data, and attributes that the deployment descriptor file can contain.

For example, the `payara-application_11_0-0.dtd` file defines the structure of the `payara-application.xml` file. The DTD files for the Payara Server deployment descriptors are located in the `as-install/lib/dtds` directory.

The Platform Server deployment descriptor files must be readable and writable by the file owners. In each deployment descriptor file, sub-elements must be defined in the order in which they are listed under each Sub-elements heading, unless otherwise noted.

For general information about DTD files and XML, see the XML specification at `http://www.w3.org/TR/REC-xml`.

IMPORTANT	Do not edit the DTD files; their contents change only with new versions of Payara Server.

The following table lists the Azul Payara deployment descriptors and their DTD files.

Table 17. Azul Payara Deployment Descriptors and DTDs

Deployment Descriptor	DTD File	Description
`glassfish-application.xml` (deprecated) `payara-application.xml`	`glassfish-application_6_0-1.dtd` (deprecated) `payara6-application_10_0-0.dtd` (deprecated) `payara-application_11_0-0.dtd` (deprecated) `payara7-application_11_0-1.dtd`	Configures an entire Jakarta EE application (EAR file).
`glassfish-web.xml` (deprecated) `payara-web.xml`	`glassfish-web-app_3_0-1.dtd` (deprecated) `payara-web-app_4.dtd` (deprecated) `payara6-web-app_6_0-0.dtd` (deprecated) `payara-web-app_6_1-0.dtd` (deprecated) `payara7-web-app_6_1-1.dtd`	Configures a web application (WAR file).

Deployment Descriptor	DTD File	Description
`glassfish-ejb-jar.xml` (deprecated) `payara-ejb-jar.xml`	`glassfish-ejb-jar_3_1-1.dtd` (deprecated) `payara6-ejb-jar_4_0-0.dtd` (deprecated) `payara-ejb-jar_4_0-0.dtd` (deprecated) `payara7-ejb-jar_4_0-1.dtd`	Configures an enterprise bean (EJB JAR file).

`glassfish-application-client.xml` (deprecated)
 	
`payara-application-client.xml`
	`glassfish-application-client_6_0-1.dtd`
	
	`payara6-application-client_10_0-0.dtd` (deprecated)
	
	`payara-application-client_11_0-0.dtd` (deprecated)
	
	`payara7-application-client_11_0-1.dtd`
		Configures an Application Client Container (ACC) client (JAR file).

`glassfish-resources.xml` (deprecated) `payara-resources.xml`	`glassfish-resources_1_5.dtd` (deprecated) `payara-resources_1_8.dtd`	Configures application-scoped resources.

Deployment Descriptor	DTD File	Description
`sun-acc.xml` (deprecated) `glassfish-acc.xml` (deprecated) `payara-acc.xml`	`sun-application-client-container_1_2.dtd` (deprecated) `glassfish-application-client-container_1_3.dtd` (deprecated) `payara6-application-client-container_1_4-0.dtd` (deprecated) `payara-application-client-container_1_4.dtd` (deprecated) `payara7-application-client-container_1_4-1.dtd`	Configures the Application Client Container. This is more of a configuration file than a deployment descriptor. Payara Server provides a default file in the `domain-dir/config` directory. Specifying a different file is optional.

IMPORTANT

Deployment descriptors whose names start with the `glassfish-` prefix are referred as "original" deployment descriptor files from Eclipse GlassFish, since these files names and DTDs were created while supporting Eclipse GlassFish and their support has been inherited by Azul Payara.

The Payara-application.xml File

IMPORTANT

glassfish-application.xml is deprecated and support will be removed in a future release. It is recommended to use payara-application.xml instead.

The `payara-application.xml` file configures an entire Jakarta EE application (EAR file). The element hierarchy is as follows:

```
payara-application
.  web
.  .  web-uri
.  .  context-root
.  pass-by-reference
.  unique-id
.  security-role-mapping
.  .  role-name
.  .  principal-name
.  .  group-name
.  realm
.  ejb-ref
.  .  ejb-ref-name
.  .  jndi-name
.  resource-ref
.  .  res-ref-name
.  .  jndi-name
.  .  default-resource-principal
.  .  .  name
.  .  .  password
.  resource-env-ref
.  .  resource-env-ref-name
.  .  jndi-name
.  service-ref
.  .  service-ref-name
.  .  port-info
.  .  .  service-endpoint-interface
.  .  .  wsdl-port
.  .  .  .  namespaceURI
.  .  .  .  localpart
.  .  .  stub-property
.  .  .  .  name
.  .  .  .  value
.  .  .  call-property
.  .  .  .  name
.  .  .  .  value
.  .  .  message-security-binding
.  .  .  .  message-security
.  .  .  .  .  message
.  .  .  .  .  .  java-method
.  .  .  .  .  .  .  method-name
.  .  .  .  .  .  .  method-params
.  .  .  .  .  .  .  .  method-param
.  .  .  .  .  .  operation-name
.  .  .  .  .  request-protection
.  .  .  .  .  response-protection
.  .  call-property
.  .  .  name
.  .  .  value
.  .  wsdl-override
.  .  service-impl-class
.  .  service-qname
.  .  .  namespaceURI
.  .  .  localpart
.  message-destination-ref
.  .  message-destination-ref-name
.  .  jndi-name
.  message-destination
.  .  message-destination-name
```

```
.  .  jndi-name
.  archive-name
.  compatibility
.  keep-state
.  version-identifier
.  classloading-delegate
.  enable-implicit-cdi
.  whitelist-package
.  scanning-exclude
.  scanning-include
```

sample `payara-application.xml` *file:*

```
<!DOCTYPE payara-application PUBLIC "-//Payara.fish//DTD
Payara Application Server 7 Jakarta EE Application 11 Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-application_11-1.dtd">
<payara-application>
   <unique-id>67488732739338240</unique-id>
</payara-application>
```

The `glassfish-web.xml` and `payara-web.xml` Files

The `glassfish-web.xml` file configures a web application (WAR file). It is an original Eclipse GlassFish deployment descriptor.

The `payara-web.xml` is a matching Azul Payara variant which is recommended to be used whenever possible.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

Their element hierarchy is as follows:

```
payara-web-app
.  context-root
.  security-role-mapping
.  .  role-name
.  .  principal-name
.  .  group-name
.  servlet
.  .  servlet-name
.  .  principal-name
.  .  webservice-endpoint
.  .  .  port-component-name
.  .  .  endpoint-address-uri
.  .  .  login-config
.  .  .  .  auth-method
.  .  .  message-security-binding
```

```
. . . . message-security
. . . . . message
. . . . . . java-method
. . . . . . . method-name
. . . . . . . method-params
. . . . . . . . method-param
. . . . . . operation-name
. . . . . request-protection
. . . . . response-protection
. . . transport-guarantee
. . . service-qname
. . . tie-class
. . . servlet-impl-class
. . . debugging-enabled
. . . property (with attributes)
. . . . description
. idempotent-url-pattern
. session-config
. . session-manager
. . . manager-properties
. . . . property (with attributes)
. . . . . description
. . . store-properties
. . . . property (with attributes)
. . . . . description
. . session-properties
. . . property (with attributes)
. . . . description
. . cookie-properties
. . . property (with attributes)
. . . . description
. ejb-ref
. . ejb-ref-name
. . jndi-name
. resource-ref
. . res-ref-name
. . jndi-name
. . default-resource-principal
. . . name
. . . password
. resource-env-ref
. . resource-env-ref-name
. . jndi-name
. service-ref
. . service-ref-name
. . port-info
. . . service-endpoint-interface
. . . wsdl-port
. . . . namespaceURI
. . . . localpart
. . . stub-property
. . . . name
. . . . value
. . . call-property
. . . . name
. . . . value
. . . message-security-binding
. . . . message-security
. . . . . message
. . . . . . java-method
```

```
. . . . . . . method-name
. . . . . . . method-params
. . . . . . . . method-param
. . . . . . operation-name
. . . . . request-protection
. . . . . response-protection
. . call-property
. . . name
. . . value
. . wsdl-override
. . service-impl-class
. . service-qname
. . . namespaceURI
. . . localpart
. message-destination-ref
. . message-destination-ref-name
. . jndi-name
. cache
. . cache-helper
. . . property (with attributes)
. . . . description
. . default-helper
. . . property (with attributes)
. . . . description
. . property (with attributes)
. . . description
. . cache-mapping
. . . servlet-name
. . . url-pattern
. . . cache-helper-ref
. . . dispatcher
. . . timeout
. . . refresh-field
. . . http-method
. . . key-field
. . . constraint-field
. . . . constraint-field-value
. class-loader
. . property (with attributes)
. . . description
. jsp-config
. locale-charset-info
. . locale-charset-map
. . parameter-encoding
. parameter-encoding
. property (with attributes)
. . description
. valve
. message-destination
. . message-destination-name
. . jndi-name
. webservice-description
. . webservice-description-name
. . wsdl-publish-location
. keep-state
. version-identifier
. jaxrs-roles-allowed-enabled
. scanning-exclude
. scanning-include
```

```
.  whitelist-package
```

A sample `payara-web.xml` *file:*

```
<!DOCTYPE payara-web-app PUBLIC "-//Payara.fish//DTD Payara Server 7 Servlet 6.1
Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-web-app_6_1-1.dtd">
<payara-web-app>
    <session-config>
        <session-manager/>
    </session-config>
    <resource-ref>
        <res-ref-name>mail/Session</res-ref-name>
        <jndi-name>mail/Session</jndi-name>
    </resource-ref>
    <jsp-config/>
</payara-web-app>
```

The `payara-ejb-jar.xml` File

IMPORTANT	glassfish-ejb-jar.xml is deprecated and support will be removed in a future release. It is recommended to use payara-ejb-jar.xml instead.

The `payara-ejb-jar.xml` file configures an enterprise bean (EJB JAR file). The element hierarchy is as follows:

```
payara-ejb-jar
.  security-role-mapping
.  .  role-name
.  .  principal-name
.  .  group-name
.  enterprise-beans
.  .  name
.  .  unique-id
.  .  ejb
.  .  .  ejb-name
.  .  .  jndi-name
.  .  .  ejb-ref
.  .  .  .  ejb-ref-name
.  .  .  .  jndi-name
.  .  .  resource-ref
.  .  .  .  res-ref-name
.  .  .  .  jndi-name
.  .  .  .  default-resource-principal
.  .  .  .  .  name
.  .  .  .  .  password
         clustered-bean
.  .  .  clustered-key-name
.  .  .  clustered-lock-type
```

```
. . . clustered-attach-postconstruct
. . . clustered-detach-predestroy
. . . resource-env-ref
. . . . resource-env-ref-name
. . . . jndi-name
. . . service-ref
. . . . service-ref-name
. . . . port-info
. . . . . service-endpoint-interface
. . . . . wsdl-port
. . . . . . namespaceURI
. . . . . . localpart
. . . . . stub-property
. . . . . . name
. . . . . . value
. . . . . call-property
. . . . . . name
. . . . . . value
. . . . . message-security-binding
. . . . . . message-security
. . . . . . . message
. . . . . . . . java-method
. . . . . . . . . method-name
. . . . . . . . . method-params
. . . . . . . . . . method-param
. . . . . . . . operation-name
. . . . . . . . request-protection
. . . . . . . . response-protection
. . . . call-property
. . . . . name
. . . . . value
. . . . wsdl-override
. . . . service-impl-class
. . . . service-qname
. . . . . namespaceURI
. . . . . localpart
. . . message-destination-ref
. . . . message-destination-ref-name
. . . . jndi-name
. . . pass-by-reference
. . . principal
. . . . name
. . . mdb-connection-factory
. . . . jndi-name
. . . . default-resource-principal
. . . . . name
. . . . . password
. . . jms-durable-subscription-name
. . . jms-max-messages-load
. . . ior-security-config
. . . . transport-config
. . . . . integrity
. . . . . confidentiality
. . . . . establish-trust-in-target
. . . . . establish-trust-in-client
. . . . as-context
. . . . . auth-method
. . . . . realm
. . . . . required
. . . . sas-context
```

```
. . . . . caller-propagation
. . . is-read-only-bean
. . . refresh-period-in-seconds
. . . commit-option
. . . cmt-timeout-in-seconds
. . . use-thread-pool-id
. . . gen-classes
. . . . remote-impl
. . . . local-impl
. . . . remote-home-impl
. . . . local-home-impl
. . . bean-pool
. . . . steady-pool-size
. . . . resize-quantity
. . . . max-pool-size
. . . . pool-idle-timeout-in-seconds
. . . bean-cache
. . . . max-cache-size
. . . . resize-quantity
. . . . is-cache-overflow-allowed
. . . . cache-idle-timeout-in-seconds
. . . . removal-timeout-in-seconds
. . . . victim-selection-policy
. . . mdb-resource-adapter
. . . . resource-adapter-mid
. . . . activation-config
. . . . . description
. . . . . activation-config-property
. . . . . . activation-config-property-name
. . . . . . activation-config-property-value
. . . webservice-endpoint
. . . . port-component-name
. . . . endpoint-address-uri
. . . . login-config
. . . . . auth-method
. . . . . realm
. . . . message-security-binding
. . . . . message-security
. . . . . . message
. . . . . . . java-method
. . . . . . . . method-name
. . . . . . . . method-params
. . . . . . . . . method-param
. . . . . . . operation-name
. . . . . . request-protection
. . . . . . response-protection
. . . . transport-guarantee
. . . . service-qname
. . . . tie-class
. . . . servlet-impl-class
. . . . debugging-enabled
. . . . property (with subelements)
. . . . . name
. . . . . value
. . . flush-at-end-of-method
. . . . method
. . . . . description
. . . . . ejb-name
. . . . . method-name
. . . . . method-intf
```

```
. . . . . method-params
. . . . . . method-param
. . . checkpointed-methods
. . . checkpoint-at-end-of-method
. . . . method
. . . . . description
. . . . . ejb-name
. . . . . method-name
. . . . . method-intf
. . . . . method-params
. . . . . . method-param
. . . per-request-load-balancing
. . pm-descriptors
. . message-destination
. . . message-destination-name
. . . jndi-name
. . webservice-description
. . . webservice-description-name
. . . wsdl-publish-location
. . property (with sub-elements)
. . . name
. . . value
. compatibility
. disable-nonportable-jndi-names
. keep-state
. version-identifier
  webservice-default-login-config
```

NOTE

If any configuration information for an enterprise bean is not specified in the `payara-ejb-jar.xml` file, it defaults to a corresponding setting in the EJB container if an equivalency exists.

A sample `payara-ejb-jar.xml` file:

```
<!DOCTYPE payara-ejb-jar PUBLIC "-//Payara.fish//
DTD Payara Application Server 7 EJB 4.0 Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-ejb-jar_4_0-1.dtd">
<payara-ejb-jar>
    <enterprise-beans>
        <ejb>
            <ejb-name>CustomerEJB</ejb-name>
            <jndi-name>customer</jndi-name>
            <bean-pool>
                <steady-pool-size>10</steady-pool-size>
                <resize-quantity>10</resize-quantity>
                <max-pool-size>100</max-pool-size>
                <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
            </bean-pool>
            <bean-cache>
                <max-cache-size>100</max-cache-size>
                <resize-quantity>10</resize-quantity>
                <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
                <victim-selection-policy>LRU</victim-selection-policy>
            </bean-cache>
```

```
        </ejb>
    </enterprise-beans>
    <keep-state>true</keep-state>
</payara-ejb-jar>
```

The `payara-application-client.xml` File

IMPORTANT	glassfish-application-client.xml is deprecated and support will be removed in a future release. It is recommended to use payara-application-client.xml instead.

The `payara-application-client.xml` file configures an Application Client Container (ACC) client (JAR file). The element hierarchy is as follows:

```
payara-application-client
.  ejb-ref
.  .  ejb-ref-name
.  .  jndi-name
.  resource-ref
.  .  res-ref-name
.  .  jndi-name
.  .  default-resource-principal
.  .  .  name
.  .  .  password
.  resource-env-ref
.  .  resource-env-ref-name
.  .  jndi-name
.  service-ref
.  .  service-ref-name
.  .  port-info
.  .  .  service-endpoint-interface
.  .  .  wsdl-port
.  .  .  .  namespaceURI
.  .  .  .  localpart
.  .  .  stub-property
.  .  .  .  name
.  .  .  .  value
.  .  .  call-property
.  .  .  .  name
.  .  .  .  value
.  .  .  message-security-binding
.  .  .  .  message-security
.  .  .  .  .  message
.  .  .  .  .  .  java-method
.  .  .  .  .  .  .  method-name
.  .  .  .  .  .  .  method-params
.  .  .  .  .  .  .  .  method-param
.  .  .  .  .  .  operation-name
.  .  .  .  .  request-protection
.  .  .  .  .  response-protection
.  .  call-property
.  .  .  name
.  .  .  value
```

```
.  .  wsdl-override
.  .  service-impl-class
.  .  service-qname
.  .  .  namespaceURI
.  .  .  localpart
.  message-destination-ref
.  .  message-destination-ref-name
.  .  jndi-name
.  message-destination
.  .  message-destination-name
.  .  jndi-name
.  java-web-start-access
.  .  context-root
.  .  eligible
.  .  vendor
.  .  jnlp-doc
.  version-identifier
```

A sample `payara-application-client.xml` *file:*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-application-client PUBLIC "-//Payara.fish//DTD Payara Application
Server 7 Jakarta EE Application Client 11//EN" "{payaraApplicationClientDtd}">
<payara-application-client>
  <message-destination-ref>
    <message-destination-ref-name>ClientQueue</message-destination-ref-name>
    <jndi-name>jms/security_mdb_OutQueue</jndi-name>
  </message-destination-ref>
</payara-application-client>
```

The `glassfish-resources.xml` and `payara-resources.xml` Files

IMPORTANT	glassfish-resources.xml is deprecated and support will be removed in a future release. It is recommended to use payara-resources.xml instead.

The `glassfish-resources.xml` file configures application-scoped resources. It is an original Eclipse GlassFish deployment descriptor.

The `payara-resources.xml` is a matching Azul Payara variant that supports additional elements.

Their element hierarchy is as follows:

```
resources
.  custom-resource
.  .  description
.  .  property (with attributes)
```

```
.  .  .  description
.  external-jndi-resource
.  .  description
.  .  property (with attributes)
.  .  .  description
.  jdbc-resource
.  .  description
.  .  property (with attributes)
.  .  .  description
.  mail-resource
.  .  description
.  .  property (with attributes)
.  .  .  description
.  admin-object-resource
.  .  description
.  .  property (with attributes)
.  .  .  description
.  connector-resource
.  .  description
.  .  property (with attributes)
.  .  .  description
.  resource-adapter-config
.  .  property (with attributes)
.  .  .  description
.  jdbc-connection-pool
.  .  description
.  .  log-jdbc-calls
.  .  property (with attributes)
.  .  .  description
.  connector-connection-pool
.  .  description
.  .  security-map
.  .  .  principal
.  .  .  user-group
.  .  .  backend-principal
.  .  property (with attributes)
.  .  .  description
.  work-security-map
.  .  description
.  .  principal-map
.  .  group-map
.  managed-executor-service (*)
.  managed-scheduled-executor-service (*)
.  managed-thread-factory (*)
```

IMPORTANT | The elements marked as `(*)` are only supported in the payara-resources.xml deployment descriptor variant.

1. A sample `payara-resources.xml`

```
<!DOCTYPE resources PUBLIC "-//Payara.fish//DTD Payara Server {release-id} Resource
Definitions//EN" "{payaraResourcesDtd}">
<resources>
    <!-- Connects to MySql database called 'test' on a server called 'database' -->
    <jdbc-connection-pool name="jdbc/testConnPool"
```

```
                            res-type="javax.sql.DataSource"
                            datasource-
classname="com.mysql.jdbc.jdbc2.optional.MysqlDataSource"
                            pool-resize-quantity="2"
                            max-pool-size="32"
                            steady-pool-size="8">
        <property name="URL" value="jdbc:mysql://database:3306/test"/>
        <property name="User" value="root"/>
        <property name="Password" value="password"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" jndi-name="jdbc/testDS" object-type="user" pool-
name="jdbc/testConnPool">
        <description>Test DataSource jdbc/testDS for Connection Pool
jdbc/testConnPool</description>
    </jdbc-resource>
</resources>
```

The Sun-acc.xml File

The `sun-acc.xml` file configures the Application Client Container. This is more of a configuration file than a deployment descriptor.

Payara Server provides a default file in the `domain-dir/config` directory. Specifying a different file is optional. The element hierarchy is as follows:

```
client-container
.  target-server
.  .  description
.  .  security
.  .  .  ssl
.  .  .  cert-db
.  auth-realm
.  .  property (with attributes)
.  client-credential
.  .  property (with attributes)
.  log-service
.  .  property (with attributes)
.  message-security-config
.  .  provider-config
.  .  .  request-policy
.  .  .  response-policy
.  .  .  property (with attributes)
.  property (with attributes)
```

Deployment Descriptor Precedence Order

When multiple vendor deployment descriptors are present in the same module (including deployment descriptors of the original **Eclipse GlassFish** and of the old **Sun Application Server**), the descriptors are processed with the following precedence order in default mode:

```
Azul Payara > Eclipse GlassFish Server > Sun Application Server
```

Elements of the Azul Payara Deployment Descriptors

This appendix describes the elements of the Azul Payara deployment descriptors.

`activation-config`

Specifies an activation configuration, which includes the runtime configuration properties of the message-driven bean in its operational environment. For example, this can include information about the name of a physical JMS destination. Matches and overrides the `activation-config` element in the `ejb-jar.xml` file.

Superelements

mdb-resource-adapter (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `activation-config` element.

Table 18. `activation-config` subelements

Element	Required	Description
description	zero or one	Specifies a text description of the activation configuration.
activation-config-property	one or more	Specifies an activation configuration property.

`activation-config-property`

Specifies the name and value of an activation configuration property.

Superelements

activation-config (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `activation-config-property` element.

Table 19. `activation-config-property` subelements

Element	Required	Description
activation-config-property-name	only one	Specifies the name of an activation configuration property.
activation-config-property-value	only one	Specifies the value of an activation configuration property.

`activation-config-property-name`

Specifies the name of an activation configuration property.

Superelements

activation-config-property (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`activation-config-property-value`

Specifies the value of an activation configuration property.

Superelements

activation-config-property (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`admin-object-resource`

Defines an administered object for an inbound resource adapter.

Superelements

resources (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `admin-object-resource` element.

Table 20. `admin-object-resource` Subelements

Element	Required	Description
description	zero or one	Contains a text description of this element.
property-with-attributes	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the `admin-object-resource` element.

Table 21. `admin-object-resource` Attributes

Attribute	Default	Description
`jndi-name`	none	Specifies the JNDI name for the resource.
`res-type`	none	Specifies the fully qualified type of the resource.
`res-adapter`	none	Specifies the name of the inbound resource adapter.
`object-type`	`user`	(optional) Defines the type of the resource. Allowed values are: • `system-all` - A system resource for all server instances and the domain application server. • `system-admin` - A system resource only for the domain application server. • `system-instance` - A system resource for all server instances only. • `user` - A user resource.
`enabled`	`true`	(optional) Determines whether this resource is enabled at runtime.

Properties

Properties of the `admin-object-resource` element are the names of setter methods

of the class referenced by the `adminobject-class` of the `ra.xml` file.

Some of the property names can be specified in the `adminobjectType` element.

`as-context`

Specifies the authentication mechanism used to authenticate the client.

Superelements

ior-security-config (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `as-context` element.

Table 22. `as-context` Subelements

Element	Required	Description
auth-method	only one	Specifies the authentication method. The only supported value is `USERNAME_PASSWORD`.
realm	only one	Specifies the realm in which the user is authenticated.
required	only one	Specifies whether the authentication method specified in the `auth-method` element must be used for client authentication.

`archive-name`

Specifies the name of the archive file. The value of the `archive-name` element is used to derive the default application name when `display-name` is not present in the `application.xml` file.

The default application name is the `archive-name` value minus the file extension. For example, if `archive-name` is `foo.ear`, the default application name is `foo`.

Superelements

payara-application (<code>glassfish-application.xml</code> (deprecated) and `payara-application.xml</code>)

Subelements

none - contains data

`auth-method`

Specifies the authentication method.

If the parent element is as-context, the only supported value is `USERNAME_PASSWORD`.

If the parent element is login-config, specifies the authentication mechanism for the web service endpoint. As a prerequisite to gaining access to any web resources protected by an authorization constraint, a user must be authenticated using the configured mechanism.

Superelements

login-config (`glassfish-web.xml` (deprecated) and `payara-web.xml`), as-context (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`auth-realm`

JAAS is available on the ACC. Defines the optional configuration for a JAAS authentication realm. Authentication realms require provider-specific properties, which vary depending on what a particular implementation needs. For more information about how to define realms, see "realm-configuration" in the Payara Server Application Development section.

Superelements

client-container (`sun-acc.xml`)

Subelements

The following table describes subelements for the `auth-realm` element.

Table 23. `auth-realm` subelement

Element	Required	Description
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

`Attributes`

The following table describes attributes for the `auth-realm` element.

Table 24. `auth-realm` attributes

Attribute	Default	Description
`name`	none	Defines the name of this realm.
`classname`	none	Defines the Jakarta class which implements this realm.

Example

Here is an example of the default file realm:

```
<auth-realm name="file"
   classname="com.sun.enterprise.security.auth.realm.file.FileRealm">
   <property name="file" value="domain-dir/config/keyfile"/>
   <property name="jaas-context" value="fileRealm"/>
</auth-realm>
```

Which properties an `auth-realm` element uses depends on the value of the `auth-realm` element's `name` attribute. The file realm uses `file` and `jaas-context` properties. Other realms use different properties. See "realm-configuration" in the Payara Server Application Development section.

`backend-principal`

Specifies the user name and password required by the Enterprise Information System (EIS).

Superelements

security-map (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

none

Attributes

The following table describes attributes for the `backend-principal` element.

Table 25. `backend-principal` Attributes

Attribute	Default	Description
`user-name`	none	Specifies the user name required by the EIS.
`password`	none	(optional) Specifies the password required by the EIS, if any.

`bean-cache`

Specifies the entity bean cache properties. Used for entity beans and stateful session beans.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `bean-cache` element.

Table 26. `bean-cache` Subelements

Element	Required	Description
max-cache-size	zero or one	Specifies the maximum number of beans allowable in cache.
cache-idle-timeout-in-seconds	zero or one	Specifies the maximum time that a stateful session bean or entity bean is allowed to be idle in cache before being passivated. Default value is 10 minutes (600 seconds).
removal-timeout-in-seconds	zero or one	Specifies the amount of time a bean remains before being removed. If `removal-timeout-in-seconds` is less than `idle-timeout`, the bean is removed without being passivated.

Element	Required	Description
resize-quantity	zero or one	Specifies the number of beans to be created if the pool is empty (subject to the `max-pool-size` limit). Values are from 0 to MAX_INTEGER.
victim-selection-policy	zero or one	Specifies the algorithm that must be used by the container to pick victims. Applies only to stateful session beans.

Example

```
<bean-cache>
   <max-cache-size>100</max-cache-size>
   <cache-resize-quantity>10</cache-resize-quantity>
   <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
   <victim-selection-policy>LRU</victim-selection-policy>
      <cache-idle-timeout-in-seconds>600</cache-idle-timeout-in-seconds>
   <removal-timeout-in-seconds>5400</removal-timeout-in-seconds>
</bean-cache>
```

`bean-pool`

Specifies the pool properties of stateless session beans, entity beans, and message-driven bean.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `bean-pool` element.

Table 27. `bean-pool` Subelements

Element	Required	Description
steady-pool-size	zero or one	Specifies the initial and minimum number of beans maintained in the pool. Default is 32.
resize-quantity	zero or one	Specifies the number of beans to be created if the pool is empty (subject to the `max-pool-size` limit). Values are from 0 to MAX_INTEGER.

Element	Required	Description
max-pool-size	zero or one	Specifies the maximum number of beans in the pool. Values are from 0 to MAX_INTEGER. Default is to the EJB container value or 60.
pool-idle-timeout-in-seconds	zero or one	Specifies the maximum time that a bean is allowed to be idle in the pool. After this time, the bean is removed. This is a hint to the server. Default time is 600 seconds (10 minutes).

Example

```
<bean-pool>
   <steady-pool-size>10</steady-pool-size>
   <resize-quantity>10</resize-quantity>
   <max-pool-size>100</max-pool-size>
   <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
</bean-pool>
```

`cache`

Configures caching for web application components.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `cache` element.

Table 28. `cache` Subelements

Element	Required	Description
cache-helper	zero or more	Specifies a custom class that implements the CacheHelper interface.
default-helper	zero or one	Allows you to change the properties of the default, built-in cache-helper class.
property-with-attributes	zero or more	Specifies a cache property, which has a name and a value.

Element	Required	Description
cache-mapping	zero or more	Maps a URL pattern or a servlet name to its cacheability constraints.

Attributes

The following table describes attributes for the `cache` element.

Table 29. `cache` Attributes

Attribute	Default	Description
`max-entries`	`4096`	(optional) Specifies the maximum number of entries the cache can contain. Must be a positive integer.
`timeout-in-seconds`	`30`	(optional) Specifies the maximum amount of time in seconds that an entry can remain in the cache after it is created or refreshed. Can be overridden by a timeout element.
`enabled`	`true`	(optional) Determines whether servlet and JSP caching is enabled.

Properties

The following table describes properties for the `cache` element.

Table 30. `cache` Properties

Property	Default	Description
`cacheClassName`	`com.sun.appserv.web.cache.LruCache`	Specifies the fully qualified name of the class that implements the cache functionality. See cache-class-names for possible values.
`MultiLRUSegmentSize`	`4096`	Specifies the number of entries in a segment of the cache table that should have its own LRU (least recently used) list. Applicable only if `cacheClassName` is set to `com.sun.appserv.web.cache.MultiLruCache`.

Property	Default	Description
`MaxSize`	unlimited; `Long.MAX_VALUE`	Specifies an upper bound on the cache memory size in bytes (KB or MB units). Example values are `32 KB` or `2 MB`. Applicable only if `cacheClassName` is set to `com.sun.appserv.web.cache.BoundedMultiLruCache`.

Cache Class Names

The following table lists possible values of the `cacheClassName` property.

Table 31. `cacheClassName` Values

Value	Description
`com.sun.appserv.web.cache.LruCache`	A bounded cache with an LRU (least recently used) cache replacement policy.
`com.sun.appserv.web.cache.BaseCache`	An unbounded cache suitable if the maximum number of entries is known.
`com.sun.appserv.web.cache.MultiLruCache`	A cache suitable for a large number of entries (>4096). Uses the `MultiLRUSegmentSize` property.
`com.sun.appserv.web.cache.BoundedMultiLruCache`	A cache suitable for limiting the cache size by memory rather than number of entries. Uses the `MaxSize` property.

`cache-helper`

Specifies a class that implements the com.sun.appserv.web.cache.CacheHelper interface.

Superelements

cache (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `cache-helper` element.

Table 32. `cache-helper` Subelements

Element
Required
Description
property-with-attributes
zero or more
Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `cache-helper` element.

Table 33. `cache-helper` Attributes

Attribute	Default	Description
`name`	`default`	Specifies a unique name for the helper class, which is referenced in the cache-mapping element.
`class-name`	none	Specifies the fully qualified class name of the cache helper, which must implement the com.sun.appserv.web.CacheHelper interface.

`cache-helper-ref`

Specifies the `name` of the cache-helper used by the parent cache-mapping element.

Superelements

cache-mapping (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none - contains data

`cache-idle-timeout-in-seconds`

Specifies the maximum time that a bean can remain idle in the cache. After this amount of time, the container can passivate this bean. A value of `0` specifies that beans never become candidates for passivation. Default is 600.

Applies to stateful session beans and entity beans.

Superelements

bean-cache (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`cache-mapping`

Maps a URL pattern or a servlet name to its cacheability constraints.

Superelements

cache (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `cache-mapping` element.

Table 34. `cache-mapping` Subelements

Element	Required	Description
servlet-name	requires one `servlet-name` or `url-pattern`	Contains the name of a servlet.
url-pattern	requires one `servlet-name` or `url-pattern`	Contains a servlet URL pattern for which caching is enabled.
cache-helper-ref	required if `dispatcher`, `timeout`, `refresh-field`, `http-method`, `key-field`, and `constraint-field` are not used	Contains the `name` of the cache-helper used by the parent `cache-mapping` element.

Element	Required	Description
dispatcher	zero or one if `cache-helper-ref` is not used	Contains a comma-separated list of `RequestDispatcher` methods for which caching is enabled.
timeout	zero or one if `cache-helper-ref` is not used	Contains the cache-mapping specific maximum amount of time in seconds that an entry can remain in the cache after it is created or refreshed.
refresh-field	zero or one if `cache-helper-ref` is not used	Specifies a field that gives the application component a programmatic way to refresh a cached entry.
http-method	zero or more if `cache-helper-ref` is not used	Contains an HTTP method that is eligible for caching.
key-field	zero or more if `cache-helper-ref` is not used	Specifies a component of the key used to look up and extract cache entries.
constraint-field	zero or more if `cache-helper-ref` is not used	Specifies a cacheability constraint for the given `url-pattern` or `servlet-name`.

`call-property`

Specifies JAX-RPC property values that can be set on a `javax.xml.rpc.Call` object before it is returned to the web service client. The property names can be any properties supported by the JAX-RPC `Call` implementation.

Superelements

port-info, service-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-`

`application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

The following table describes subelements for the `call-property` element.

Table 35. `call-property` subelements

Element	Required	Description
name	only one	Specifies the name of the entity.
value	only one	Specifies the value of the entity.

`caller-propagation`

Specifies whether the target accepts propagated caller identities. The values are `NONE`, `SUPPORTED`, or `REQUIRED`.

Superelements

sas-context (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`cert-db`

Not implemented. Included for backward compatibility only. Attribute values are ignored.

Superelements

security (`sun-acc.xml`)

Subelements

none

Attributes

The following table describes attributes for the `cert-db` element.

Table 36. `cert-db` attributes

Attribute	Default	Description
`path`	none	Specifies the absolute path of the certificate database.
`password`	none	Specifies the password to access the certificate database.

`checkpoint-at-end-of-method`

Specifies that the stateful session bean state is checkpointed, or persisted, after the specified methods are executed. The `availability-enabled` attribute of the parent ejb element must be set to `true`.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `checkpoint-at-end-of-method` element.

Table 37. `checkpoint-at-end-of-method` Subelements

Element	Required	Description
method	one or more	Specifies a bean method.

`class-loader`

Configures the class loader for the web module.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `class-loader` element.

Table 38. `class-loader` Subelements

Element	Required	Description
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `class-loader` element.

Table 39. `class-loader` Attributes

Attribute	Default	Description
`extra-class-path`	null	(optional) Specifies a colon or semicolon separated list of additional classpaths for this web module. Paths can be absolute or relative to the web module's root, for example: `extra-class-path="WEB-INF/lib/extra/extra.jar"`
`delegate`	`true`	(optional) If `true`, the web module follows the standard class loader delegation model and delegates to its parent class loader first before looking in the local class loader. You must set this to `true` for a web module that accesses EJB components or that acts as a web service client or endpoint. If `false`, the web module follows the delegation model specified in the Servlet specification and looks in its class loader before looking in the parent class loader. It's safe to set this to `false` only for a web module that does not interact with any other modules. For a number of packages, including `java. and javax.`, symbol resolution is always delegated to the parent class loader regardless of the delegate setting. This prevents applications from overriding core Java runtime classes or changing the API versions of specifications that are part of the Jakarta EE platform.
`dynamic-reload-interval`	+	(optional) Not implemented. Included for backward compatibility with previous Oracle Web Server versions.

NOTE	If the `delegate` attribute is set to `false`, the class loader delegation behavior complies with the Servlet 6.0 specification, section 10.7.2. If set

> to its default value of `true`, classes and resources residing in container-wide library JAR files are loaded in preference to classes and resources packaged within the WAR file. Portable programs that use this element should not be packaged with any classes or interfaces that are a part of the Jakarta EE specification. The behavior of a program that includes such classes or interfaces in its WAR file is undefined.

Properties

The following table describes properties for the `class-loader` element.

Table 40. `class-loader` Properties

Property	Default	Description
`ignoreHiddenJarFiles`	`false`	If `true`, specifies that all JAR and ZIP files in the `WEB-INF/lib` directory that start with a period (`.`) are ignored by the class loader.

`classloading-delegate`

With this option its possible to enable/disable **class loading delegation**. This allows deployed application to use libraries included on them, overriding the versions included on the server.

For more information about how class delegation can be configured on Payara Server, see the enhanced-classloading section.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`), payara-application (`glassfish-application.xml` (deprecated) and `payara-application.xml`)

Subelements

none

`client-container`

Defines the Payara Server specific configuration for the application client container.

This is the root element; there can only be one `client-container` element in a `sun-acc.xml` file. See the-sun-acc.xml-file.

Superelements

none

Subelements

The following table describes subelements for the `client-container` element.

Table 41. `client-container` Subelements

Element	Required	Description
target-server	one or more	Specifies the IIOP listener for the target server. Also specifies IIOP endpoints used for load balancing. If the Payara Server instance on which the application client is deployed participates in a cluster, Payara Server finds all currently active IIOP endpoints in the cluster automatically. However, a client should have at least two endpoints specified for bootstrapping purposes, in case one of the endpoints has failed. A listener or endpoint is in the form `host:port`, where the host is an IP address or host name, and the port specifies the port number.
auth-realm	zero or one	Specifies the optional configuration for JAAS authentication realm.
client-credential	zero or one	Specifies the default client credential that is sent to the server.
log-service	zero or one	Specifies the default log file and the severity level of the message.
message-security-config	zero or more	Specifies configurations for message security providers.
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `client-container` element.

Table 42. `client-container` Attributes

Attribute	Default	Description
`send-password`	`true`	If `true`, specifies that client authentication credentials must be sent to the server. Without authentication credentials, all access to protected EJB components results in exceptions.

`client-credential`

Default client credentials that are sent to the server. If this element is present, the credentials are automatically sent to the server, without prompting the user for the user name and password on the client side.

Superelements

client-container (`sun-acc.xml`)

Subelements

The following table describes subelements for the `client-credential` element.

Table 43. `client-credential` subelement

Element	Required	Description
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `client-credential` element.

Table 44. `client-credential` attributes

Attribute	Default	Description
`user-name`	none	The user name used to authenticate the Application client container.

Attribute	Default	Description
`password`	none	The password used to authenticate the Application client container.
`realm`	default realm for the domain	(optional) The realm (specified by name) where credentials are to be resolved.

`clustered-attach-postconstruct`

Describes whether to call a `@PostConstruct` event listener each time the corresponding singleton bean is created on a different node. This setup will result in multiple calls. Valid values are `true` or `false`.

The default value is `true`.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none

`clustered-bean`

Describes whether this bean should be a "Clustered Singleton". Can be applied only to singleton EJBs.

Valid values are `true` or `false`. The default value is `false`.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none

`clustered-detach-predestroy`

Whether to call a `@PreDestroy` event listener when a singleton EJB bean is destroyed on an instance while still being available on another. This setup will result in multiple calls.

Valid values are `true` or `false`. The default value is `true`.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none

`clustered-key-name`

The key used for replication of clustered singletons beans. Applies to a singleton EJBs when the `clustered-bean` element is set to `true`.

WARNING	This element is optional. If not set, the default key is the value of the `ejb-name` element.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none

`clustered-lock-type`

Describes the type of distributed locking to be performed for a clustered singleton bean:

- For EJB beans, only `INHERIT` and `LOCK_NONE` are valid.
- For CDI beans, valid values are `LOCK` and `INHERIT`, which is equivalent to using `LOCK_NONE`.

The default value is `INHERIT`

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none

`cmt-timeout-in-seconds`

Overrides the Transaction Timeout setting of the Transaction Service for an individual bean. The default value, `0`, specifies that the default Transaction Service timeout is used. If positive, this value is used for all methods in the bean that start a new container-managed transaction. This value is not used if the bean joins a client transaction.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`commit-option`

Specifies the commit option used on transaction completion. Valid values for Payara Server are `B` or `C`. Default value is `B`. Applies to entity beans.

NOTE	Commit option A is not supported for this Payara Server release.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`compatibility`

Specifies the Payara Server release with which to be backward compatible in terms of JAR visibility requirements for applications. The current allowed value is `v2`, which refers to older runtimes from where Payara Server is based from. Starting in Java EE 6, the Java EE specification imposes stricter requirements than Java EE 5 did on which JAR files can be visible to various modules within an EAR file. Setting this element to `v2` removes these Java EE 6 and later restrictions.

Superelements

payara-application (`glassfish-application.xml` (deprecated) and `payara-`

`application.xml`), payara-ejb-jar (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`confidentiality`

Specifies if the target supports privacy-protected messages. The values are `NONE`, `SUPPORTED`, or `REQUIRED`.

Superelements

transport-config (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`connector-connection-pool`

Defines a connector connection pool.

Superelements

resources (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `connector-connection-pool` element.

Table 45. `connector-connection-pool` Subelements

Element	Required	Description
description	zero or one	Contains a text description of this element.
security-map	zero or more	Maps the principal received during servlet or EJB authentication to the credentials accepted by the EIS.
property-with-attributes	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the `connector-connection-pool` element. Changing the following attributes requires a server restart or the redeployment or disabling and re-enabling of applications that refer to the resource: `resource-adapter-name`, `connection-definition-name`, `transaction-support`, `associate-with-thread`, `lazy-connection-association`, and `lazy-connection-enlistment`.

Table 46. `connector-connection-pool` Attributes

Attribute	Default	Description
`name`	none	Specifies the name of the connection pool. A connector-resource element's `pool-name` attribute refers to this `name`.
`resource-adapter-name`	none	Specifies the name of the deployed connector module or application. If no name is specified during deployment, the name of the `.rar` file is used. If the resource adapter is embedded in an application, then it is `app_name#rar_name`.
`connection-definition-name`	none	Specifies a unique name, identifying a resource adapter's `connection-definition` element in the `ra.xml` file. This is usually the `connectionfactory-interface` of the `connection-definition` element.
`steady-pool-size`	`8`	(optional) Specifies the initial and minimum number of connections maintained in the pool.
`max-pool-size`	`32`	(optional) Specifies the maximum number of connections that can be created to satisfy client requests.
`max-wait-time-in-millis`	`60000`	(optional) Specifies the amount of time, in milliseconds, that the caller is willing to wait for a connection. If `0`, the caller is blocked indefinitely until a resource is available or an error occurs.

Attribute	Default	Description
`pool-resize-quantity`	`2`	(optional) Specifies the number of idle connections to be destroyed if the existing number of connections is above the `steady-pool-size` (subject to the `max-pool-size` limit). This is enforced periodically at the `idle-timeout-in-seconds` interval. An idle connection is one that has not been used for a period of `idle-timeout-in-seconds`. When the pool size reaches `steady-pool-size`, connection removal stops.
`idle-timeout-in-seconds`	`300`	(optional) Specifies the maximum time that a connection can remain idle in the pool. After this amount of time, the pool can close this connection.
`fail-all-connections`	`false`	(optional) If `true`, closes all connections in the pool if a single validation check fails.
`transaction-support`	none	(optional) Specifies the transaction support for this connection pool. Overrides the transaction support defined in the resource adapter in a downward compatible way: supports a transaction level lower than or equal to the resource adapter's, but not higher. Allowed values in descending order are: • `XATransaction` - Supports distributed transactions. • `LocalTransaction` - Supports local transactions only. • `NoTransaction` - No transaction support.
`is-connection-validation-required`	`false`	(optional) Specifies whether connections have to be validated before being given to the application. If a resource's validation fails, it is destroyed, and a new resource is created and returned.

Attribute	Default	Description
`validate-atmost-once-period-in-seconds`	`0`	Specifies the time interval within which a connection is validated at most once. Minimizes the number of validation calls. A value of zero allows unlimited validation calls.
`connection-leak-timeout-in-seconds`	`0`	Detects potential connection leaks by the application. A connection that is not returned back to the pool by the application within the specified period is assumed to be potentially leaking, and a stack trace of the caller is logged. A zero value disables leak detection. A nonzero value enables leak tracing.
`connection-leak-reclaim`	`false`	If `true`, the pool will reclaim a connection after `connection-leak-timeout-in-seconds` occurs.
`connection-creation-retry-attempts`	`0`	Specifies the number of attempts to create a new connection.
`connection-creation-retry-interval-in-seconds`	`10`	Specifies the time interval between attempts to create a connection when `connection-creation-retry-attempts` is greater than `0`.
`lazy-connection-enlistment`	`false`	If `true`, a connection is not enlisted in a transaction until it is used. If `false`, any connection object available to a transaction is enlisted in the transaction.
`lazy-connection-association`	`false`	If `true`, a physical connection is not associated with a logical connection until it is used. If `false`, a physical connection is associated with a logical connection even before it is used.

Attribute	Default	Description
`associate-with-thread`	`false`	If `true`, allows connections to be saved as `ThreadLocal` in the calling thread. Connections get reclaimed only when the calling thread dies or when the calling thread is not in use and the pool has run out of connections. If `false`, the thread must obtain a connection from the pool each time the thread requires a connection. This attribute associates connections with a thread such that when the same thread is in need of connections, it can reuse the connections already associated with that thread. In this case, the overhead of getting connections from the pool is avoided. However, when this value is set to `true`, you should verify that the value of the `max-pool-size` attribute is comparable to the `max-thread-pool-size` attribute of the associated thread pool. If the `max-thread-pool-size` value is much higher than the `max-pool-size` value, a lot of time is spent associating connections with a new thread after dissociating them from an older one. Use this attribute in cases where the thread pool should reuse connections to avoid this overhead.
`match-connections`	`true`	If `true`, enables connection matching. You can set to `false` if connections are homogeneous.
`max-connection-usage-count`	`0`	Specifies the number of times a connections is reused by the pool, after which it is closed. A zero value disables this feature.
`ping`	`false`	(optional) Specifies whether to ping the pool during pool creation or reconfiguration to identify and warn of any erroneous attribute values.
`pooling`	`true`	(optional) If `false`, disables connection pooling.

Properties

Most properties of the `connector-connection-pool` element are the names of setter methods of the `managedconnectionfactory-class` element in the `ra.xml` file. Properties of the `connector-connection-pool` element override the `ManagedConnectionFactory` JavaBean configuration settings.

All but the last four properties in the following table are `connector-connection-pool` properties of `jmsra`, the resource adapter used to communicate with the Open Message Queue software.

Changes to `connector-connection-pool` properties require a server restart.

Table 47. `connector-connection-pool` Properties

Property	Default	Description
`AddressList`	none	Specifies a list of host/port combinations of the Message Queue software. For JMS resources of the Type `jakarta.jms.TopicConnectionFactory` or `jakarta.jms.QueueConnectionFactory`.
`ClientId`	none	Specifies the JMS Client Identifier to be associated with a `Connection` created using the `createTopicConnection` method of the `TopicConnectionFactory` class. For JMS resources of the Type `jakarta.jms.TopicConnectionFactory`. Durable subscription names are unique and only valid within the scope of a client identifier. To create or reactivate a durable subscriber, the connection must have a valid client identifier. The JMS specification ensures that client identifiers are unique and that a given client identifier is allowed to be used by only one active connection at a time.

Property	Default	Description
`UserName`	`guest`	Specifies the user name for connecting to the Message Queue software. For JMS resources of the Type `jakarta.jms.TopicConnectionFactory` or `jakarta.jms.QueueConnectionFactory`.
`Password`	`guest`	Specifies the password for connecting to the Message Queue software. For JMS resources of the Type `jakarta.jms.TopicConnectionFactory` or `jakarta.jms.QueueConnectionFactory`.
`ReconnectAttempts`	`6`	Specifies the number of attempts to connect (or reconnect) for each address in the `imqAddressList` before the client runtime moves on to try the next address in the list. A value of `-1` indicates that the number of reconnect attempts is unlimited (the client runtime attempts to connect to the first address until it succeeds).
`ReconnectInterval`	`30000`	Specifies the interval between reconnect attempts in milliseconds. This applies to attempts on each address in the `imqAddressList` and on successive addresses in the list. If too short, this time interval does not give a broker time to recover. If too long, the reconnect might represent an unacceptable delay.
`ReconnectEnabled`	`false`	If `true`, specifies that the client runtime attempts to reconnect to a message server (or the list of addresses in `imqAddressList`) when a connection is lost.
`AddressListBehavior`	`priority`	Specifies whether connection attempts are in the order of addresses in the `imqAddressList` attribute (`priority`) or in a random order (`random`). If many clients are attempting a connection using the same connection factory, use a random order to prevent them from all being connected to the same address.

Property	Default	Description
`AddressListIterations`	`-1`	Specifies the number of times the client runtime iterates through the `imqAddressList` in an effort to establish (or reestablish) a connection. A value of `-1` indicates that the number of attempts is unlimited.

`connector-resource`

Defines the connection factory object of a specific connection definition in a connector (resource adapter).

Superelements

resources (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `connector-resource` element.

Table 48. `connector-resource` Subelements

Element	Required	Description
description	zero or one	Contains a text description of this element.
property-with-attributes	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the `connector-resource` element.

Table 49. `connector-resource` Attributes

Attribute	Default	Description
`jndi-name`	none	Specifies the JNDI name for the resource.
`pool-name`	none	Specifies the `name` of the associated connector-connection-pool.

Attribute	Default	Description
`object-type`	`user`	(optional) Defines the type of the resource. Allowed values are: • `system-all` - A system resource for all server instances and the domain application server. • `system-admin` - A system resource only for the domain application server. • `system-instance` - A system resource for all server instances only. • `user` - A user resource.
`enabled`	`true`	(optional) Determines whether this resource is enabled at runtime.

`constraint-field`

Specifies a cacheability constraint for the given url-pattern or servlet-name.

All `constraint-field` constraints must pass for a response to be cached. If there are `value` constraints, at least one of them must pass.

Superelements

cache-mapping (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `constraint-field` element.

Table 50. `constraint-field` Subelements

Element	Required	Description
constraint-field-value	zero or more	Contains a value to be matched to the input parameter value.

Attributes

The following table describes attributes for the `constraint-field` element.

Table 51. `constraint-field` Attributes

Attribute	Default	Description
`name`	none	Specifies the input parameter name.
`scope`	`request.parameter`	(optional) Specifies the scope from which the input parameter is retrieved. Allowed values are `context.attribute`, `request.header`, `request.parameter`, `request.cookie`, `request.attribute`, and `session.attribute`.
`cache-on-match`	`true`	(optional) If `true`, caches the response if matching succeeds. Overrides the same attribute in a constraint-field-value subelement.
`cache-on-match-failure`	`false`	(optional) If `true`, caches the response if matching fails. Overrides the same attribute in a constraint-field-value subelement.

`constraint-field-value`

Specifies a value to be matched to the input parameter value. The matching is case sensitive. For example:

```
<value match-expr="in-range">1-60</value>
```

Superelements

constraint-field (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none - contains data

Attributes

The following table describes attributes for the `constraint-field-value` element.

Table 52. `constraint-field-value` Attributes

Attribute	Default	Description
`match-expr`	`equals`	(optional) Specifies the type of comparison performed with the value. Allowed values are `equals`, `not-equals`, `greater`, `lesser`, and `in-range`. If `match-expr` is `greater` or `lesser`, the value must be a number. If `match-expr` is `in-range`, the value must be of the form `n1-n2`, where `n1` and `n2` are numbers.
`cache-on-match`	`true`	(optional) If `true`, caches the response if matching succeeds.
`cache-on-match-failure`	`false`	(optional) If `true`, caches the response if matching fails.

`context-root`

Contains the web context root for the application or web application that was packaged as a WAR file. Overrides the corresponding element in the `application.xml` or `web.xml` file.

If the parent element is `java-web-start-access`, this element contains the context root for the Java Web Start enabled application client module. If none is specified, a default is generated; see java-web-start-access.

If you are setting up load balancing, web module context roots must be unique within a server instance.

Superelements

web (`glassfish-application.xml` (deprecated) and `payara-application.xml`), payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`), java-web-start-access (`glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`cookie-properties`

Specifies session cookie properties.

NOTE	If cookie settings are defined declaratively in the `web.xml` file, the cookie properties defined here take precedence. If cookie settings are defined programmatically using `jakarta.servlet.SessionCookieConfig` methods, those cookie settings take precedence over the cookie properties defined here.

Superelements

session-config (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `cookie-properties` element.

Table 53. `cookie-properties` Subelements

Element	Required	Description
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Properties

The following table describes properties for the `cookie-properties` element.

Table 54. `cookie-properties` Properties

Property	Default	Description
`cookieName`	none	Specifies the cookie name.
`cookiePath`	Context path at which the web module is installed.	Specifies the pathname that is set when the cookie is created. The browser sends the cookie if the pathname for the request contains this pathname. If set to `/` (slash), the browser sends cookies to all URLs served by Payara Server. You can set the path to a narrower mapping to limit the request URLs to which the browser sends cookies.

Property	Default	Description
`cookieMaxAgeSeconds`	none	Specifies the expiration time (in seconds) after which the browser expires the cookie. If this is unset, the cookie doesn't expire.
`cookieDomain`	(unset)	Specifies the domain for which the cookie is valid.
`cookieComment`	none	Specifies the comment that identifies the session tracking cookie in the cookie file.
`cookieSecure`	`dynamic`	Sets the `Secure` attribute of any `JSESSIONID` cookies associated with the web application. Allowed values are as follows: • `true` — Sets `Secure` to `true`. • `false` — Sets `Secure` to `false`. • `dynamic` — The `JSESSIONID` cookie inherits the `Secure` setting of the request that initiated the session. To set the `Secure` attribute of a `JSESSIONIDSSO` cookie, use the `ssoCookieSecure virtual-server` property. For details, see "`create-virtual-server`".
`cookieHttpOnly`	none	Specifies that the cookie is marked HTTP only. Allowed values are `true` or `false`.
`cookieSameSite`	none	Users can set up the `SameSite` Cookie attribute for the corresponding web application. Can be one of `Strict`, `Lax` or `None`. For example: `<property name="cookieSameSite" value="Strict" />`

`custom-resource`

Defines a custom resource, which specifies a custom server-wide resource object factory. Such object factories implement the `javax.naming.spi.ObjectFactory` interface.

Superelements

resources (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `custom-resource` element.

Table 55. `custom-resource` Subelements

Element	Required	Description
description	zero or one	Contains a text description of this element.
property-with-attributes	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the `custom-resource` element.

Table 56. `custom-resource` Attributes

Attribute	Default	Description
`jndi-name`	none	Specifies the JNDI name for the resource.
`res-type`	none	Specifies the fully qualified type of the resource.
`factory-class`	none	Specifies the fully qualified name of the user-written factory class, which implements `javax.naming.spi.ObjectFactory`.
`object-type`	`user`	(optional) Defines the type of the resource. Allowed values are: • `system-all` - A system resource for all server instances and the domain application server. • `system-admin` - A system resource only for the domain application server. • `system-instance` - A system resource for all server instances only. • `user` - A user resource.

Attribute	Default	Description
`enabled`	`true`	(optional) Determines whether this resource is enabled at runtime.

`debugging-enabled`

Specifies whether the debugging servlet is enabled for this web service endpoint.

Allowed values are `true` (the default) and `false`.

Superelements

webservice-endpoint (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`default-helper`

Passes property values to the built-in `default` cache-helper class.

Superelements

cache (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `default-helper` element.

Table 57. `default-helper` Subelements

Element	Required	Description
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Properties

The following table describes properties for the `default-helper` element.

Table 58. `default-helper` Properties

Property	Default	Description
`cacheKeyGeneratorAttrName`	Uses the built-in `default` cache-helper key generation, which concatenates the servlet path with key-field values, if any.	The caching engine looks in the `ServletContext` for an attribute with a name equal to the value specified for this property to determine whether a customized CacheKeyGenerator implementation is used. An application can provide a customized key generator rather than using the `default` helper. See "the-cachekeygenerator-interface" in the Payara Server Application Development section.

`default-resource-principal`

Specifies the default principal (user) for the resource.

If this element is used in conjunction with a JMS Connection Factory resource, the `name` and `password` subelements must be valid entries in the Open Message Queue broker user repository.

Superelements

resource-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`); mdb-connection-factory (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `default-resource-principal` element.

Table 59. `default-resource-principal` Subelements

Element	Required	Description
name	only one	Specifies the default resource principal name used to sign on to a resource manager.
password	only one	Specifies password of the default resource principal.

`description`

Specifies a text description of the containing element.

Superelements

property-with-attributes, valve (`glassfish-web.xml` (deprecated) and `payara-web.xml`); activation-config, method (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`); target-server (`sun-acc.xml`); admin-object-resource, connector-connection-pool, connector-resource, custom-resource, external-jndi-resource, jdbc-connection-pool, "`jdbc-resource`", mail-resource, property-with-attributes, resource-adapter-config (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

none - contains data

`disable-nonportable-jndi-names`

Because the Jakarta Enterprise Beans specification defines portable EJB JNDI names, there is less need for Payara Server specific JNDI names. By default, Payara Server specific default JNDI names are applied automatically for backward compatibility. To disable Payara Server specific JNDI names for an EJB module, set the value of this element to `true`. The default is `false`.

Superelements

payara-ejb-jar (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`dispatcher`

Specifies a comma-separated list of `RequestDispatcher` methods for which caching is enabled on the target resource. Valid values are `REQUEST`, `FORWARD`, `INCLUDE`, and `ERROR` . If this element is not specified, the default is `REQUEST`. See `6.2.5` of the Servlet 6.0 specification for more information.

Superelements

cache-mapping (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none - contains data

`ejb`

Defines runtime properties for a single enterprise bean within the application. The subelements listed below apply to particular enterprise beans as follows:

- All types of beans: `ejb-name`, `ejb-ref`, `resource-ref`, `resource-env-ref`, `ior-security-config`, `gen-classes`, `jndi-name`, `use-thread-pool-id`, `message-destination-ref`, `pass-by-reference`, `service-ref`
- Stateless session beans: `bean-pool`, `webservice-endpoint`
- Stateful session beans: `bean-cache`, `webservice-endpoint`, `checkpoint-at-end-of-method`
- Entity beans: `commit-option`, `bean-cache`, `bean-pool`, `is-read-only-bean`, `refresh-period-in-seconds`, `flush-at-end-of-method`
- Message-driven beans: `mdb-resource-adapter`, `mdb-connection-factory`, `jms-durable-subscription-name`, `jms-max-messages-load`, `bean-pool`

Superelements

enterprise-beans (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `ejb` element.

Table 60. `ejb` Subelements

Element	Required	Description
ejb-name	only one	Matches the `ejb-name` in the corresponding `ejb-jar.xml` file.

Element	Required	Description
jndi-name	zero or more	Specifies the absolute `jndi-name`.
ejb-ref	zero or more	Maps the absolute JNDI name to the `ejb-ref` element in the corresponding Jakarta EE XML file.
resource-ref	zero or more	Maps the absolute JNDI name to the `resource-ref` in the corresponding Jakarta EE XML file.
resource-env-ref	zero or more	Maps the absolute JNDI name to the `resource-env-ref` in the corresponding Jakarta EE XML file.
service-ref	zero or more	Specifies runtime settings for a web service reference.
message-destination-ref	zero or more	Specifies the name of a physical message destination.
pass-by-reference	zero or one	Specifies the passing method used by an enterprise bean calling a remote interface method in another bean that is colocated within the same process.
principal	zero or one	Specifies the principal (user) name in an enterprise bean that has the `run-as` role specified.
mdb-connection-factory	zero or one	Specifies the connection factory associated with a message-driven bean.
jms-durable-subscription-name	zero or one	Specifies the durable subscription associated with a message-driven bean.
jms-max-messages-load	zero or one	Specifies the maximum number of messages to load into a Jakarta Messaging session at one time for a message-driven bean to serve. The default is 1.
ior-security-config	zero or one	Specifies the security information for the IOR.
is-read-only-bean	zero or one	Specifies that this entity bean is read-only.
refresh-period-in-seconds	zero or one	Specifies the rate at which a read-only-bean must be refreshed from the data source.
commit-option	zero or one	Has valid values of B or C. Default value is B.
cmt-timeout-in-seconds	zero or one	Overrides the Transaction Timeout setting of the Transaction Service for an individual bean.

Element	Required	Description
use-thread-pool-id	zero or one	Specifies the thread pool from which threads are selected for remote invocations of this bean.
gen-classes	zero or one	Specifies all the generated class names for a bean.
bean-pool	zero or one	Specifies the bean pool properties. Used for stateless session beans, entity beans, and message-driven beans.
bean-cache	zero or one	Specifies the bean cache properties. Used only for stateful session beans and entity beans.
mdb-resource-adapter	zero or one	Specifies runtime configuration information for a message-driven bean.
webservice-endpoint	zero or more	Specifies information about a web service endpoint.
flush-at-end-of-method	zero or one	Specifies the methods that force a database flush after execution. Used for entity beans.
checkpoint-at-end-of-method	zero or one	Specifies that the stateful session bean state is checkpointed, or persisted, after the specified methods are executed. The `availability-enabled` attribute must be set to `true`.
per-request-load-balancing	zero or one	Specifies the per-request load balancing behavior of EJB 2.x and 3.x remote client invocations on a stateless session bean.
clustered-attach-postconstruct	zero or one	Used for clustered singletons. Defines whether the bean's `@PostConstruct` event is called each time it is created on different nodes.
clustered-bean	zero or one	Defines whether the bean is treated as a clustered singleton.
clustered-detach-predestroy	zero or one	Used for clustered singletons. Defines whether the bean's `@PreDestroy` event is called when the bean is destroyed on a node while available on another.

Element	Required	Description
clustered-key-name	zero or one	Used for clustered singletons. Defines the name of the key used for the replication mechanism of the bean across the Data Grid.
clustered-lock-type	zero or one	Used for clustered singletons. Defines the type of distributed lock to be performed on the bean.

Attributes

The following table describes attributes for the `ejb` element.

Table 61. `ejb` Attributes

Attribute	Default	Description
`availability-enabled`	`false`	(optional) If set to `true`, and if availability is enabled in the EJB container, high-availability features apply to this bean if it is a stateful session bean.

Example

```
<ejb>
   <ejb-name>CustomerEJB</ejb-name>
   <jndi-name>customer</jndi-name>
   <resource-ref>
      <res-ref-name>jdbc/SimpleBank</res-ref-name>
      <jndi-name>jdbc/__default</jndi-name>
   </resource-ref>
   <is-read-only-bean>false</is-read-only-bean>
   <commit-option>B</commit-option>
   <bean-pool>
      <steady-pool-size>10</steady-pool-size>
      <resize-quantity>10</resize-quantity>
      <max-pool-size>100</max-pool-size>
      <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
   </bean-pool>
   <bean-cache>
      <max-cache-size>100</max-cache-size>
      <resize-quantity>10</resize-quantity>
      <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
      <victim-selection-policy>LRU</victim-selection-policy>
   </bean-cache>
</ejb>
```

`ejb-name`

In the `payara-ejb-jar.xml` file, matches the `ejb-name` in the corresponding `ejb-`

`jar.xml` file. The name must be unique among the names of the enterprise beans in the same EJB JAR file.

There is no architected relationship between the `ejb-name` in the deployment descriptor and the JNDI name that the deployer assigns to the EJB component's home.

Superelements

ejb, method (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`ejb-ref`

Maps the `ejb-ref-name` in the corresponding Jakarta EE deployment descriptor file `ejb-ref` entry to the absolute `jndi-name` of a resource.

The `ejb-ref` element is used for the declaration of a reference to an EJB's home. Applies to session beans or entity beans.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`), ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`), payara-application-client (`glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

The following table describes subelements for the `ejb-ref` element.

Table 62. `ejb-ref` Subelements

Element	Required	Description
ejb-ref-name	only one	Specifies the `ejb-ref-name` in the corresponding Jakarta EE deployment descriptor file `ejb-ref` entry.
jndi-name	only one	Specifies the absolute `jndi-name` of a resource.

`ejb-ref-name`

Specifies the `ejb-ref-name` in the corresponding Jakarta EE deployment descriptor file `ejb-ref` entry.

Superelements

ejb-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`eligible`

Specifies whether the application client module is eligible to be Java Web Start enabled. Allowed values are `true` (the default) and `false`.

Superelements

java-web-start-access (`glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`enable-implicit-cdi`

In a **WAR** file, it is possible to set the property `bean-discovery-mode` equal to `none` to turn off implicit scanning of the archive for bean defining annotations, as defined by the CDI 1.1 specification.

The default value of this setting is defined as `annotated` in the specification, so the archive is scanned for any bean-defining annotations, which can cause unwanted side effects.

In the `payara-application.xml` deployment descriptor for an EAR file, the property `enable-implicit-cdi` can be set to `false` to achieve the same goal for all modules inside the EAR assembly. The default value is `true`, in line with the default value for

WAR files.

If implicit CDI scanning causes problems for an EAR assembly, the value `false` will disable implicit CDI scanning for all CDI modules inside the EAR assembly:

```
<payara-application>
  <enable-implicit-cdi>false</enable-implicit-cdi>
</payara-application>
```

The default behavior of the admin console is for the **Implicit CDI** checkbox to be enabled, but this will **not** override the application configuration.

IMPORTANT	When implicit CDI is configured by using either the `enable-implicit-cdi` property in the `payara-application.xml` or the attribute `bean-discovery-mode="none"` from the `beans.xml` file in a WAR, the admin console checkbox *is always ignored*.

Superelements

payara-application

Subelements

none - contains data

`endpoint-address-uri`

Specifies the relative path combined with the web server root to form the fully qualified endpoint address for a web service endpoint. This is a required element for EJB endpoints and an optional element for servlet endpoints.

For servlet endpoints, this value is relative to the web application context root. For EJB endpoints, the URI is relative to root of the web server (the first portion of the URI is a context root). The context root portion must not conflict with the context root of any web application deployed to the same web server.

In all cases, this value must be a fixed pattern (no "*' allowed).

If the web service endpoint is a servlet that implements only a single endpoint and has only one `url-pattern`, it is not necessary to set this value, because the web container derives it from the `web.xml` file.

Superelements

webservice-endpoint (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

Example

If the web server is listening at `http://localhost:8080`, the following `endpoint-address-uri`:

```
<endpoint-address-uri>StockQuoteService/StockQuotePort</endpoint-address-uri>
```

results in the following target endpoint address:

```
http://localhost:8080/StockQuoteService/StockQuotePort
```

`enterprise-beans`

Specifies all the runtime properties for an EJB JAR file in the application.

Superelements

payara-ejb-jar (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `enterprise-beans` element.

Table 63. `enterprise-beans` Subelements

Element	Required	Description
name	zero or one	Specifies the name string.

Element	Required	Description
unique-id	zero or one	Specifies a unique system identifier. This data is automatically generated and updated at deployment/redeployment. Do not specify or edit this value.
ejb	zero or more	Defines runtime properties for a single enterprise bean within the application.
message-destination	zero or more	Specifies the name of a logical message destination.
webservice-description	zero or more	Specifies a name and optional publish location for a web service.
property-with-subelements	zero or more	Specifies a property or a variable.

Example

```
<enterprise-beans>
 <ejb>
     <ejb-name>CustomerEJB</ejb-name>
     <jndi-name>customer</jndi-name>
     <resource-ref>
         <res-ref-name>jdbc/SimpleBank</res-ref-name>
         <jndi-name>jdbc/__default</jndi-name>
     </resource-ref>
     <is-read-only-bean>false</is-read-only-bean>
     <commit-option>B</commit-option>
     <bean-pool>
         <steady-pool-size>10</steady-pool-size>
        <resize-quantity>10</resize-quantity>
         <max-pool-size>100</max-pool-size>
         <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
     </bean-pool>
     <bean-cache>
         <max-cache-size>100</max-cache-size>
         <resize-quantity>10</resize-quantity>
         <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
         <victim-selection-policy>LRU</victim-selection-policy>
     </bean-cache>
 </ejb>
</enterprise-beans>
```

Properties

The following table describes properties for the `enterprise-beans` element.

Table 64. `enterprise-beans` Properties

Property	Default	Description
`default-role-mapping`	`false`	With this property, you can set whether to enable the default group to role mappings for your application's security settings. This element is set up as a `property` element with a `Boolean` value attribute like this: `<property name="default-role-mapping" value="true"><description>Default role mapping</description></property>` Enabling the default group to role mappings will cause all named groups in the module's linked **security realm** to be mapped to a role of the same name. This will save you the time of having to redefine the same roles and map them to the realm groups each time they are modified.

`establish-trust-in-client`

Specifies if the target is capable of authenticating a client. The values are `NONE`, `SUPPORTED`, or `REQUIRED`.

Superelements

transport-config (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`establish-trust-in-target`

Specifies if the target is capable of authenticating to a client. The values are `NONE`, `SUPPORTED`, or `REQUIRED`.

Superelements

transport-config (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`external-jndi-resource`

Defines a resource that resides in an external JNDI repository. For example, a generic Java object could be stored in an LDAP server. An external JNDI factory must implement the javax.naming.spi.InitialContextFactory interface.

Superelements

resources (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `external-jndi-resource` element.

Table 65. `external-jndi-resource` *Subelements*

Element	Required	Description
description	zero or one	Contains a text description of this element.
property-with-attributes	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the `external-jndi-resource` element.

Table 66. `external-jndi-resource` *Attributes*

Attribute	Default	Description
`jndi-name`	none	Specifies the JNDI name for the resource.
`jndi-lookup-name`	none	Specifies the JNDI lookup name for the resource.
`res-type`	none	Specifies the fully qualified type of the resource.
`factory-class`	none	Specifies the fully qualified name of the factory class, which implements javax.naming.spi.InitialContextFactory .

Attribute	Default	Description
`object-type`	`user`	(optional) Defines the type of the resource. Allowed values are: • `system-all` - A system resource for all server instances and the domain application server. • `system-admin` - A system resource only for the domain application server. • `system-instance` - A system resource for all server instances only. • `user` - A user resource.
`enabled`	`true`	(optional) Determines whether this resource is enabled at runtime.

`flush-at-end-of-method`

Specifies the methods that force a database flush after execution. Applicable to entity beans.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `flush-at-end-of-method` element.

Table 67. `flush-at-end-of-method` Subelements

Element	Required	Description
method	one or more	Specifies a bean method.

`gen-classes`

Specifies all the generated class names for a bean.

NOTE	This value is automatically generated by the server at deployment or redeployment time. Do not specify it or change it after deployment.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `gen-classes` element.

Table 68. `gen-classes` Subelements

Element	Required	Description
remote-impl	zero or one	Specifies the fully-qualified class name of the generated `EJBObject` impl class.
local-impl	zero or one	Specifies the fully-qualified class name of the generated `EJBLocalObject` impl class.
remote-home-impl	zero or one	Specifies the fully-qualified class name of the generated `EJBHome` impl class.
local-home-impl	zero or one	Specifies the fully-qualified class name of the generated `EJBLocalHome` impl class.

`payara-application`

Defines the Payara Server specific configuration for an application. This is the root element; there can only be one `payara-application` element in a `payara-application.xml` file. See the-payara-application.xml-file.

Superelements

none

Subelements

The following table describes subelements for the `payara-application` element.

Table 69. `payara-application` Subelements

Element	Required	Description
web	zero or more	Specifies the application's web tier configuration.
pass-by-reference	zero or one	Determines whether EJB modules use pass-by-value or pass-by-reference semantics.

Element	Required	Description
unique-id	zero or one	Contains the unique ID for the application.
security-role-mapping	zero or more	Maps a role in the corresponding Jakarta EE XML file to a user or group.
realm	zero or one	Specifies an authentication realm.
ejb-ref	zero or more	Maps the absolute JNDI name to the `ejb-ref` in the corresponding Jakarta EE XML file.
resource-ref	zero or more	Maps the absolute JNDI name to the `resource-ref` in the corresponding Jakarta EE XML file.
resource-env-ref	zero or more	Maps the absolute JNDI name to the `resource-env-ref` in the corresponding Jakarta EE XML file.
service-ref	zero or more	Specifies runtime settings for a web service reference.
message-destination-ref	zero or more	Specifies the name of a physical message destination.
message-destination	zero or more	Specifies the name of a logical message destination.
archive-name	zero or one	Specifies the name of the archive file.
compatibility	zero or one	Specifies the Payara Server release with which to be backward compatible in terms of JAR visibility requirements for applications.
keep-state	zero or one	Retains web sessions, stateful session bean instances, and persistently created EJB timers across redeployments.
version-identifier	zero or one	Contains version information for an application.
classloading-delegate	zero or one	Configures classloading delegation for the enterprise application.
enable-implicit-cdi	zero or one	Enables implicit CDI bean scanning for the enterprise archive.
scanning-exclude-and-scanning-include	zero or one	Includes third party JAR files for CDI scanning using pattern matching.

Element	Required	Description
scanning-exclude-and-scanning-include	zero or one	Excludes third party JAR files for CDI scanning using pattern matching.
whitelist-package	zero or one	Marks packages to be whitelisted for extreme classloading isolation.

Properties

The following table describes properties for the `payara-application` element.

Table 70. `payara-application` Properties

Property	Default	Description

`default-role-mapping`	`false`	With this property, you can set whether to enable the default group to role mappings for your application's security settings. This element is set up as a `property` element with a `Boolean` value attribute like this:
		
		`<property name="default-role-mapping" value="true">`
		`  <description>Enable default group to role mapping</description>`
		`</property>`
		
		Enabling the default group to role mappings will cause all named groups in the application's linked **security realm** to be mapped to a role of the same name. This will save you the time of having to redefine the same roles and map them to the realm groups each time they are modified.
		
		This will have the same effect as executing the following `asadmin` command:
		
		`asadmin set configs.config.server-config.security-service.activate-default-principal-to-role-mapping=true`
		
		Except its effect will only limit itself to the application instead of all applications deployed on the server.
		
		WARNING When set using this property, this configuration will ALWAYS take effect and if set in the `payara-web.xml` or `payara-ejb-jar.xml` deployment descriptors, it will be ignored.

Payara-Application-Client

Defines the Payara Server specific configuration for an application client. This is the root element; there can only be one `payara-application-client` element in a

`payara-application-client.xml` file. See the-payara-application-client.xml-file.

Superelements

none

Subelements

The following table describes subelements for the `payara-application-client` element.

Table 71. `payara-application-client` subelements

Element	Required	Description
ejb-ref	zero or more	Maps the absolute JNDI name to the `ejb-ref` in the corresponding Jakarta EE XML file.
resource-ref	zero or more	Maps the absolute JNDI name to the `resource-ref` in the corresponding Jakarta EE XML file.
resource-env-ref	zero or more	Maps the absolute JNDI name to the `resource-env-ref` in the corresponding Jakarta EE XML file.
service-ref	zero or more	Specifies runtime settings for a web service reference.
message-destination-ref	zero or more	Specifies the name of a physical message destination.
message-destination	zero or more	Specifies the name of a logical message destination.
java-web-start-access	zero or one	Specifies changes to default Java Web Start parameters.
version-identifier	zero or one	Contains version information for an application client.

`payara-ejb-jar`

Defines the Payara Server specific configuration for an EJB JAR file. This is the root element; there can only be one `payara-ejb-jar` element in a `payara-ejb-jar.xml` file.

Superelements

none

Subelements

The following table describes subelements for the `payara-ejb-jar` element.

Table 72. `payara-ejb-jar` Subelements

Element	Required	Description
security-role-mapping	zero or more	Maps a role in the corresponding Jakarta EE XML file to a user or group.
enterprise-beans	only one	Describes all the runtime properties for an EJB JAR file in the application.
compatibility	zero or one	Specifies the Payara Server release with which to be backward compatible in terms of JAR visibility requirements for applications.
disable-nonportable-jndi-names	zero or one	Disables Payara Server specific JNDI names.
keep-state	zero or one	Retains stateful session bean instances and persistently created EJB timers across redeployments.
version-identifier	zero or one	Contains version information for an EJB module.
webservice-default-login-config	zero or one	Configures the default login module settings that will apply to all EJBs hosted in this archive.

`glassfish-web-app` or **`payara-web-app`**

Defines Payara Server specific configuration for a web module. This is the root element; there can only be one `glassfish-web-app`/`payara-web-app` element in a `glassfish-web.xml` (deprecated)/`payara-web.xml` file.

See the-payara-web.xml-file.

Superelements

none

Subelements

The following table describes subelements for the `payara-web-app` element.

Table 73. `payara-web-app` Subelements

Element	Required	Description
context-root	zero or one	Contains the web context root for the web module.
security-role-mapping	zero or more	Maps roles to users or groups in the currently active realm.
servlet	zero or more	Specifies a principal name for a servlet, which is used for the `run-as` role defined in `web.xml`.
idempotent-url-pattern	zero or more	Specifies a URL pattern for idempotent requests.
session-config	zero or one	Specifies session manager, session cookie, and other session-related information.
ejb-ref	zero or more	Maps the absolute JNDI name to the `ejb-ref` in the corresponding Jakarta EE XML file.
resource-ref	zero or more	Maps the absolute JNDI name to the `resource-ref` in the corresponding Jakarta EE XML file.
resource-env-ref	zero or more	Maps the absolute JNDI name to the `resource-env-ref` in the corresponding Jakarta EE XML file.
service-ref	zero or more	Specifies runtime settings for a web service reference.
message-destination-ref	zero or more	Specifies the name of a physical message destination.
cache	zero or one	Configures caching for web application components.
class-loader	zero or one	Specifies class loader configuration information.
jsp-config	zero or one	Specifies JSP configuration information.
parameter-encoding	zero or one	Determines the default request character encoding and how the web container decodes parameters from forms according to a hidden field value.
property-with-attributes	zero or more	Specifies a property, which has a name and a value.
valve	zero or more	Specifies a custom valve.
message-destination	zero or more	Specifies the name of a logical message destination.

Element	Required	Description
webservice-description	zero or more	Specifies a name and optional publish location for a web service.
keep-state	zero or one	Retains web sessions across redeployments.
version-identifier	zero or one	Contains version information for a web application.
classloading-delegate	zero or one	Configures classloading delegation for the web application.
jaxrs-roles-allowed-enabled	zero or one	Configures out of the box support for the `@RolesAllowed` annotation in a web application.
scanning-exclude-and-scanning-include	zero or one	Includes third party JAR files for CDI scanning using pattern matching.
scanning-exclude-and-scanning-include	zero or one	Excludes third party JAR files for CDI scanning using pattern matching.
whitelist-package	zero or one	Marks packages to be whitelisted for extreme classloading isolation.

Attributes

The following table describes attributes for the `payara-web-app` element.

Table 74. `payara-web-app` Attributes

Attribute	Default	Description
`error-url`	(blank)	(optional) Not implemented. Do not use.
`httpservlet-security-provider`	none	(optional) Specifies the `HttpServlet` message layer provider that the web container's servlet `auth-constraint` processing calls.

Properties

The following table describes properties for the `payara-web-app` element.

Table 75. `payara-web-app` Properties

Property	Default	Description
`allowLinking`	`false`	If `true`, resources in this web application that are symbolic links are served. You can also define this property for a virtual server. Web applications on the virtual server that do not define this property use the virtual server's value. For details, see "`create-virtual-server`(1)". **CAUTION** | Setting this property to `true` on Windows systems exposes JSP source code.

Property	Default	Description

`alternatedocroot_n`	none	Specifies an alternate document root (docroot), where n is a positive integer that allows specification of more than one. Alternate docroots allow web applications to serve requests for certain resources from outside their own docroot, based on whether those requests match one (or more) of the URI patterns of the web application's alternate docroots.
		
		If a request matches an alternate docroot's URI pattern, it is mapped to the alternate docroot by appending the request URI (minus the web application's context root) to the alternate docroot's physical location (directory). If a request matches multiple URI patterns, the alternate docroot is determined according to the following precedence order:
		
		• Exact match
		• Longest path match
		• Extension match
		
		For example, the following properties specify three alternate docroots. The URI pattern of the first alternate docroot uses an exact match, whereas the URI patterns of the second and third alternate docroots use extension and longest path prefix matches, respectively.
		
		`<payara-web-app>`
		`<property name="alternatedocroot_1" value="from=/my.jpg dir=/srv/images/jpg"/>`
		`<property name="alternatedocroot_2" value="from=*.jpg dir=/srv/images/jpg"/>`
		`<property name="alternatedocroot_3" value="from=/jpg/* dir=/src/images"/>`
		`</payara-web-app>`
		
		The `value` of each alternate docroot has two components: The first component, `from`, specifies the alternate docroot's URI pattern, and the second component, `dir`, specifies the alternate docroot's physical location (directory). Spaces are allowed in the

Property	Default	Description
`listener_n`	none	Specifies a fully qualified class name of a custom Catalina listener, where n is a positive integer that allows specification of more than one. The listener class must implement the org.apache.catalina.ContainerListener, org.apache.catalina.LifecycleListener, or org.apache.catalina.InstanceListener interface. For example: `<property name="listener_1" value="org.glassfish.extension.MyLifecycleListener "/>` You can set this property for all the web applications on a specific virtual server. For details, see "`create-virtual-server`".
`crossContextAllowed`	`true`	If `true`, allows this web application to access the contexts of other web applications using the `ServletContext.getContext()` method.
`relativeRedirectAllowed`	`false`	If `true`, allows this web application to send a relative URL to the client using `HttpServletResponse.sendRedirect()`, and instructs the web container not to translate any relative URLs to fully qualified ones.
`reuseSessionID`	`false`	If `true`, sessions generated for this web application use the session ID specified in the request.
`securePagesWithPragma`	`true`	Set this property to `false` to ensure that for this web application file downloads using SSL work properly in Internet Explorer. You can set this property for all the web applications on a specific virtual server. For details, see "`create-virtual-server`(1)".

Property	Default	Description
`singleThreadedServletPoolSize`	`5`	Specifies the maximum number of servlet instances allocated for each `SingleThreadModel` servlet in the web application.

`tempdir`	`domain-dir/generated/app-name`
	
	or
	
	`domain-dir/generated/module-name`
		Specifies a temporary directory for use by this web module. This value is used to construct the value of the `javax.servlet.context.tempdir` context attribute. Compiled JSP files are also placed in this directory.

`useResponseCTForHeaders`	`false`	If `true`, response headers are encoded using the response's charset instead of the default (UTF-8).
`default-role-mapping`	`false`	With this property, you can set whether to enable the default group to role mappings for your application's security settings. This element is set up as a `property` element with a `Boolean` value attribute like this: `<property name="default-role-mapping" value="true"> <description>Enable default group to role mapping</description> </property>` Enabling the default group to role mappings will cause all named groups in the application's linked **security realm** to be mapped to a role of the same name. This will save you the time of having to redefine the same roles and map them to the realm groups each time they are modified.

Property	Default	Description
`container-initializer-enabled`	`true`	This property configures whether to enable or disable the calling of `ServletContainerInitializer` component classes defined in JAR files bundled inside a WAR assembly. For performance considerations, you can explicitly disable the servlet container initializer by setting the `container-initializer-enabled` element to `false`. This can help solve the deployment of web applications that can suffer from conflicts with a custom bootstrapping process.
`cacheTTL`	`5000`	This property sets the time to live (TTL), in milliseconds, for the virtual server cache used when serving files. It can also be applied to all web applications on a specific virtual server. When set at the application level, this property overrides the virtual server configuration.

`group-map`

Maps an EIS group to a group defined in the Payara Server domain.

Superelements

work-security-map (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

none

Attributes

The following table describes attributes for the `group-map` element.

Table 76. `group-map` Attributes

Attribute	Default	Description
`eis-group`	none	Specifies an EIS group.
`mapped-group`	none	Specifies a group defined in the Payara Server domain.

`group-name`

Specifies a group name in the current realm.

Superelements

security-role-mapping (`glassfish-application.xml` (deprecated) and `payara-application.xml`, `glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`http-method`

Specifies an HTTP method that is eligible for caching. The default is `GET`.

Superelements

cache-mapping (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none - contains data

`idempotent-url-pattern`

Specifies a URL pattern for idempotent requests.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none

Attributes

The following table describes attributes for the `idempotent-url-pattern` element.

Table 77. `idempotent-url-pattern` Attributes

Attribute	Default	Description
`url-pattern`	none	Specifies a URL pattern, which can contain wildcards. The URL pattern must conform to the mappings specified in section 12.2 of the Servlet 6.0 specification.
`no-of-retries`	`-1`	(optional) Specifies the number of times the load balancer retries an idempotent request. A value of `-1` indicates infinite retries.

Example

The following example specifies that all requests for the URI `sun-java/*` are idempotent.

```
<idempotent-url-pattern url-pattern="sun_java/*" no-of-retries="10"/>
```

`integrity`

Specifies if the target supports integrity-protected messages. The values are `NONE`, `SUPPORTED`, or `REQUIRED`.

Superelements

transport-config (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`ior-security-config`

Specifies the security information for the interoperable object reference (IOR).

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `ior-security-config` element.

Table 78. `ior-security-config` Subelements

Element	Required	Description
transport-config	zero or one	Specifies the security information for transport.
as-context	zero or one	Specifies the authentication mechanism used to authenticate the client. If specified, it is `USERNAME_PASSWORD`.
sas-context	zero or one	Describes the sas-context fields.

Superelements

bean-cache (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

`is-read-only-bean`

Specifies that this entity bean is a read-only bean if `true`. If this element is absent, the default value of `false` is used.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`java-method`

Specifies a method.

Superelements

message (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-applcation-client.xml`)

Subelements

The following table describes subelements for the `java-method` element.

Table 79. `java-method` Subelements

Element	Required	Description
method-name	only one	Specifies a method name.
method-params	zero or one	Specifies fully qualified Java type names of method parameters.

`java-web-start-access`

Specifies changes to default Java Web Start parameters for an embedded or stand-alone application client module.

Superelements

payara-application-client (`glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

The following table describes subelements for the `java-web-start-access` element.

Table 80. `java-web-start-access` subelements

Element	Required	Description

context-root	zero or one	Contains the context root for the Java Web Start enabled application client module. If none is specified, a default is generated.
		
		The default for a web module is as follows:
		
		`http://host:port/app-name/relative-URI-to-appclient-jar`
		
		The default for a stand-alone application client module is as follows:
		
		`http://host:port/module-name`
		
		If the `module-name` is not specified during deployment, the name of the EAR or JAR file without the extension is used. If the web module is not in EAR or JAR file format, a name is generated and written to the server log.

eligible	zero or one	Specifies whether the application client module is eligible to be Java Web Start enabled. Allowed values are `true` (the default) and `false`.
vendor	zero or one	Specifies the name of the vendor as it appears in Java Web Start download and launch screens. The default value is `Application Client`.
jnlp-doc	zero or one	Specifies the name of a custom JNLP file. If none is specified, a default JNLP file is generated.

`jaxrs-roles-allowed-enabled`

Azul Payara both support using the `@RolesAllowed` annotation out of the box to secure JAX-RS resources. In some cases this may clash with existing code that interprets the same annotation using custom code.

The out-of-the-box support of `@RolesAllowed` for JAX-RS resources can be switched off by setting the `<jaxrs-roles-allowed-enabled>` element like this:

```
<jaxrs-roles-allowed-enabled>false</jaxrs-roles-allowed-enabled>
```

Superelements

payara-web-app (`glassfish-web-app.xml`)

Subelements

none

`jdbc-connection-pool`

Defines the attributes and properties that are required for creating a JDBC connection pool.

Superelements

resources (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `jdbc-connection-pool` element.

Table 81. `jdbc-connection-pool` Subelements

Element	Required	Description
description	zero or one	Contains a text description of this element.
property-with-attributes	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the `jdbc-connection-pool` element. Changing the following attributes requires a server restart or the redeployment or disabling and re-enabling of applications that refer to the resource: `datasource-classname`, `associate-with-thread`, `lazy-connection-association`, and `lazy-connection-enlistment`.

Table 82. `jdbc-connection-pool` Attributes

Attribute	Default	Description
`name`	none	Specifies the name of the connection pool. A "`jdbc-resource`" element's `pool-name` attribute refers to this `name`.
`datasource-classname`	none	(optional) Specifies the class name of the associated vendor-supplied data source. This class must implement javax.sql.DataSource, javax.sql.XADataSource, javax.sql.ConnectionPoolDatasource , or a combination.
`res-type`	none	(optional) Specifies the interface the data source class implements. The value of this attribute can be javax.sql.DataSource, javax.sql.XADataSource , javax.sql.ConnectionPoolDatasource , or java.sql.Driver. To support configuration of JDBC drivers and applications that use java.sql.Driver implementations, set this attribute to java.sql.Driver. This attribute must be specified to avoid ambiguity when a data source class implements two or more of these interfaces or when a `driver-classname` is specified. An error occurs if this attribute has a legal value and the indicated interface is not implemented by the data source class.
`driver-classname`	none	(optional) Specifies the vendor-supplied JDBC driver class name. This driver must implement the java.sql.Driver interface.
`ping`	`false`	(optional) Specifies whether to ping the pool during pool creation or reconfiguration to identify and warn of any erroneous attribute values.
`steady-pool-size`	`8`	(optional) Specifies the initial and minimum number of connections maintained in the pool.
`max-pool-size`	`32`	(optional) Specifies the maximum number of connections that can be created to satisfy client requests.

Attribute	Default	Description
`max-wait-time-in-millis`	`60000`	(optional) Specifies the amount of time, in milliseconds, that the caller is willing to wait for a connection. If `0`, the caller is blocked indefinitely until a resource is available or an error occurs.
`pool-resize-quantity`	`2`	(optional) Specifies the number of idle connections to be destroyed if the existing number of connections is above the `steady-pool-size` (subject to the `max-pool-size` limit). This is enforced periodically at the `idle-timeout-in-seconds` interval. An idle connection is one that has not been used for a period of `idle-timeout-in-seconds`. When the pool size reaches `steady-pool-size`, connection removal stops.
`idle-timeout-in-seconds`	`300`	(optional) Specifies the maximum time that a connection can remain idle in the pool. After this amount of time, the pool can close this connection. This timeout value must be kept shorter than the server side (database) timeout value to prevent the accumulation of unusable connections in the application.
`transaction-isolation-level`	default JDBC driver isolation level	(optional) Specifies the transaction isolation level on the pooled database connections. Allowed values are `read-uncommitted`, `read-committed`, `repeatable-read`, `serializable`, or `snapshot` (snapshot isolation level is only supported by Microsoft SQL Server) . Applications that change the isolation level on a pooled connection programmatically risk polluting the pool, which can lead to errors. See `is-isolation-level-guaranteed` for more details.

Attribute	Default	Description
`is-isolation-level-guaranteed`	`true`	(optional) Applicable only when `transaction-isolation-level` is explicitly set. If `true`, every connection obtained from the pool is guaranteed to have the desired isolation level. This might impact performance on some JDBC drivers. Only set this attribute to `false` if you are certain that the hosted applications do not return connections with altered isolation levels.
`is-connection-validation-required`	`false`	(optional) Specifies whether connections have to be validated before being given to the application. If a resource's validation fails, it is destroyed, and a new resource is created and returned.

Attribute	Default	Description
`connection-validation-method`	`table`	(optional) Legal values are as follows: • `auto-commit`, which uses `Connection.setAutoCommit(Connection.getAutoCommit())` • `meta-data`, which uses `Connection.getMetaData()` • `table`, which performs a query on a table specified in the `validation-table-name` attribute • `custom-validation`, which uses a user-defined validation mechanism specified by the custom implementation class in `validation-classname`. Because many JDBC drivers cache the results of `auto-commit` and `meta-data` calls, they do not always provide reliable validations. Check with the driver vendor to determine whether these calls are cached or not. The `table` must exist and be accessible, but it doesn't require any rows. Do not use an existing table that has a large number of rows or a table that is already frequently accessed.
`validation-table-name`	none	(optional) Specifies the table name to be used to perform a query to validate a connection. This parameter is mandatory if and only if `connection-validation-method` is set to `table`.

Attribute	Default	Description
`validation-classname`	none	(optional) Specifies the custom validation implementation class name. This parameter is mandatory if `connection-validation-method` is set to `custom-validation`. The classname provided must be accessible to the Payara Server. The specified class must implement the org.glassfish.api.jdbc.ConnectionValidation interface. Payara Server provides the following custom validation class templates for MSSQL, DB2, and Sybase databases. All of them implement the org.glassfish.api.jdbc.ConnectionValidation interface. • `org.glassfish.api.jdbc.MSSQLConnectionValidation` • `org.glassfish.api.jdbc.DB2ConnectionValidation` • `org.glassfish.api.jdbc.SybaseConnectionValidation`
`init-sql`	none	(optional) Specifies an SQL string to be executed whenever a connection is created (not reused) in the pool. This initializes the state of the connection.
`fail-all-connections`	`false`	(optional) If `true`, closes all connections in the pool if a single validation check fails. This parameter is mandatory if and only if `is-connection-validation-required` is set to `true`.
`non-transactional-connections`	`false`	(optional) If `true`, non-transactional connections can be made to the JDBC connection pool. These connections are not automatically enlisted with the transaction manager.

Attribute	Default	Description
`allow-non-component-callers`	`false`	(optional) If `true`, non-Java-EE components, such as servlet filters, and third party persistence managers, can use this JDBC connection pool. The returned connection is automatically enlisted with the transaction context obtained from the transaction manager. Standard Jakarta EE components can also use such pools. Connections obtained by non-component callers are not automatically closed at the end of a transaction by the container. They must be explicitly closed by the caller.
`validate-atmost-once-period-in-seconds`	`0`	(optional) Specifies the time interval within which a connection is validated at most once. Minimizes the number of validation calls. A value of zero implies that Payara Server does not attempt to minimize the number of validation requests by a connection. That is, a value of zero disables this attribute. As a result, the same connection is validated every time the application acquires the connection.
`connection-leak-timeout-in-seconds`	`0`	(optional) Detects potential connection leaks by the application. A connection that is not returned back to the pool by the application within the specified period is assumed to be potentially leaking, and a stack trace of the caller is logged. A zero value disables leak detection. A nonzero value enables leak tracing. Use this attribute along with `connection-leak-reclaim` to avoid potential connection leaks from the application.
`connection-leak-reclaim`	`false`	(optional) If `true`, the pool will reclaim a connection after `connection-leak-timeout-in-seconds` occurs.
`connection-creation-retry-attempts`	`0`	(optional) Specifies the number of attempts to create a new connection in case of a failure.

Attribute	Default	Description
`connection-creation-retry-interval-in-seconds`	`10`	(optional) Specifies the time interval between attempts to create a connection when `connection-creation-retry-attempts` is greater than `0`.
`statement-leak-timeout-in-seconds`	`0`	(optional) Detects potential statement leaks by the application. A statement that is not closed by the application within the specified period is assumed to be potentially leaking, and a stack trace of the caller is logged. A zero value disables leak detection. A nonzero value enables leak tracing. Use this attribute along with `statement-leak-reclaim` to avoid potential statement leaks from the application.
`statement-leak-reclaim`	`false`	(optional) If `true`, the reclaim of a statement after `statement-leak-timeout-in-seconds` occurs.
`statement-timeout-in-seconds`	`-1`	(optional) Sets the query timeout property of a statement to enable termination of abnormally long running queries. The default value of `-1` disables this feature. An abnormally long running JDBC query executed by an application may leave it in a hanging state unless a timeout is explicitly set on the statement. This attribute guarantees that all queries automatically time out if not completed within the specified period. When statements are created, the `queryTimeout` is set according to the value specified in this attribute. This works only when the underlying JDBC driver supports `queryTimeout` for `Statement`, `PreparedStatement`, `CallableStatement`, and `ResultSet`.

Attribute	Default	Description
`lazy-connection-enlistment`	`false`	(optional) If `true`, a connection is not enlisted in a transaction until it is used. If `false`, any connection object available to a transaction is enlisted in the transaction.
`lazy-connection-association`	`false`	(optional) If `true`, a physical connection is not associated with a logical connection until it is used. If `false`, a physical connection is associated with a logical connection even before it is used.
`associate-with-thread`	`false`	(optional) Specifies whether connections are associated with the thread to enable the thread to reuse the connections. If `true`, allows connections to be saved as `ThreadLocal` in the calling thread. Connections get reclaimed only when the calling thread dies or when the calling thread is not in use and the pool has run out of connections. If `false`, the thread must obtain a connection from the pool each time the thread requires a connection. This attribute associates connections with a thread such that when the same thread is in need of connections, it can reuse the connections already associated with that thread. In this case, the overhead of getting connections from the pool is avoided. However, when this value is set to `true`, you should verify that the value of the `max-pool-size` attribute is comparable to the `max-thread-pool-size` attribute of the associated thread pool. If the `max-thread-pool-size` value is much higher than the `max-pool-size` value, a lot of time is spent associating connections with a new thread after dissociating them from an older one. Use this attribute in cases where the thread pool should reuse connections to avoid this overhead.

Attribute	Default	Description
`match-connections`	`false`	(optional) Specifies whether a connection that is selected from the pool should be matched with the connections with certain credentials. If `true`, enables connection matching. You can set to `false` if connections are homogeneous. If the connection pool is used by applications that have multiple user credentials, `match-connections` must be `true`. The connection pool matches the request's credential with the connections in the pool and returns a matched connection for use. For new requests with different credentials, unmatched free connections are automatically purged to provide new connections to satisfy the new requests. This attribute need not be `true` if it is known that there is only one credential used by the applications and therefore the pool has homogeneous connections.
`max-connection-usage-count`	`0`	(optional) Specifies the number of times a connections is reused by the pool, after which it is closed. A zero value disables this feature. By limiting the maximum number of times a connection can be reused, you can avoid statement leaks if the application does not close statements.
`sql-trace-listeners`	none	(optional) Specifies that SQL statements executed by applications need to be traced. Helps administrators analyze the statements. Expects as a value a comma-separated list of listener implementation class names. Enables easy filtering of log messages for the SQL statements. SQL trace listeners must implement the org.glassfish.api.jdbc.SQLTraceListener interface.
`statement-cache-size`	`0`	(optional) Specifies the number of statements to be cached using the `lru` (Least Recently Used) caching mechanism. The default value of `0` disables statement caching.

Attribute	Default	Description
`pooling`	`true`	(optional) If `false`, disables connection pooling.
`wrap-jdbc-objects`	`true`	(optional) If `true`, wrapped JDBC objects are returned for `Statement`, `PreparedStatement`, `CallableStatement`, `ResultSet`, and `DatabaseMetaData`. This option ensures that `Statement.getConnection()` is the same as `DataSource.getConnection()`. Therefore, this option should be `true` when both `Statement.getConnection()` and `DataSource.getConnection()` are done. The default is `false` to avoid breaking existing applications.

Payara Server Properties

The following table describes properties for the `jdbc-connection-pool` element that are specific to Payara Server.

Table 83. `jdbc-connection-pool` Database Properties

Property	Default	Description
`dynamic-reconfiguration-wait-timeout-in-seconds`	none	Specifies the timeout for dynamic reconfiguration of the pool. In-progress connection requests must complete before this timeout expires or they must be retried. New connection requests wait for this timeout to expire before acquiring connections to the reconfigured pool. If this property exists and has a positive value, it is enabled. If this property is not set and pool reconfiguration results in pool recreation, in-progress connection requests must be retried.

Property	Default	Description
`number-of-top-queries-to-report`	`10`	Specifies the number of most frequently used queries to display. For example, the default value of `10` displays the top ten queries. This property is disabled when `jdbc-connection-pool` monitoring is set to `LOW` or `OFF`. It is enabled when `jdbc-connection-pool` monitoring is set to `HIGH` and the `sql-trace-listeners` attribute is set.
`time-to-keep-queries-in-minutes`	`5`	Specifies the time to retain queries in a cache before they are purged. This property is disabled when `jdbc-connection-pool` monitoring is set to `LOW` or `OFF`. It is enabled when `jdbc-connection-pool` monitoring is set to `HIGH` and the `sql-trace-listeners` attribute is set.

Database Properties

Most JDBC drivers allow use of standard property lists to specify the user, password, and other resource configuration information. Although properties are optional with respect to the Payara Server, some properties might be necessary for most databases. For details, see the JDBC 4.0 Standard Extension API.

When properties are specified, they are passed to the vendor's data source class (specified by the `datasource-classname` attribute) as is using setName(value) methods.

The `user` and `password` properties are used as the default principal if container managed authentication is specified and a `default-resource-principal` is not found in the application deployment descriptors.

The following table describes some common properties for the `jdbc-connection-pool` element.

Changing JDBC driver properties requires a server restart.

Table 84. `jdbc-connection-pool` Database Properties

Property	Description
`user`	Specifies the user name for connecting to the database.
`password`	Specifies the password for connecting to the database.
`databaseName`	Specifies the database for this connection pool.
`serverName`	Specifies the database server for this connection pool.
`port`	Specifies the port on which the database server listens for requests.
`networkProtocol`	Specifies the communication protocol.
`roleName`	Specifies the initial SQL role name.
`datasourceName`	Specifies an underlying XADataSource, or a ConnectionPoolDataSource if connection pooling is done.
`description`	Specifies a text description.
`url`	Specifies the URL for this connection pool. Although this is not a standard property, it is commonly used.

`jdbc-resource`

Defines a JDBC (javax.sql.DataSource) resource.

Superelements

resources (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `jdbc-resource` element.

Table 85. `jdbc-resource` Subelements

Element	Required	Description
description	zero or one	Contains a text description of this element.
property-with-attributes	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the `jdbc-resource` element.

Table 86. `jdbc-resource` Attributes

Attribute	Default	Description
`jndi-name`	none	Specifies the JNDI name for the resource.
`description`	none	(optional) Specifies a text description of this element.
`pool-name`	none	Specifies the `name` of the associated jdbc-connection-pool.
`object-type`	`user`	(optional) Defines the type of the resource. Allowed values are: • `system-all` - A system resource for all server instances and the domain application server. • `system-admin` - A system resource only for the domain application server. • `system-instance` - A system resource for all server instances only. • `user` - A user resource.
`enabled`	`true`	(optional) Determines whether this resource is enabled at runtime.

`jms-durable-subscription-name`

Specifies the durable subscription associated with a message-driven bean class. Only applies to the Jakarta Messaging Topic Destination type, and only when the message-driven bean deployment descriptor subscription durability is Durable.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`jms-max-messages-load`

Specifies the maximum number of messages to load into a Jakarta Messaging session at one time for a message-driven bean to serve. The default is 1.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`jndi-name`

Specifies the absolute `jndi-name` of a URL resource or a resource.

For entity beans and session beans, this value specifies the global JNDI name of the `EJBHome` object. It is only needed if the entity or session bean exposes a remote view.

For JMS message-driven beans, this is the JNDI name of the JMS resource from which the message-driven bean consumes JMS messages. This information is alternatively specified within the activation-config subelement of the mdb-resource-adapter element. For more information about JMS resources, see ""Using the Jakarta Messaging"" in the Payara Server Application Development section.

Superelements

ejb-ref, message-destination, resource-env-ref, resource-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`); ejb, mdb-connection-factory (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`jnlp-doc`

Contains the name of a custom JNLP file, which modifies the behavior of a Java Web Start enabled application client module. If none is specified, a default JNLP file is

generated.

The value of this element is a relative path with the following format:

```
[path-to-JAR-in-EAR!]path-to-JNLP-in-JAR
```

The default path-to-JAR-in-EAR is the current application client JAR file. For example, if the JNLP file is in the application client JAR file at `custom/myInfo.jnlp`, the element value would look like this:

```
<java-web-start-access>
   <jnlp-doc>custom/myInfo.jnlp</jnlp-doc>
</java-web-start-access>
```

If the application client is inside an EAR file, you can place the custom JNLP file inside another JAR file in the EAR. For example, if the JNLP file is in a JAR file at `other/myLib.jar`, the element value would look like this, with an exclamation point (`!`) separating the path to the JAR from the path in the JAR:

```
<java-web-start-access>
   <jnlp-doc>other/myLib.jar!custom/myInfo.jnlp</jnlp-doc>
</java-web-start-access>
```

For information about the allowed contents of a custom JNLP file, see "developing-java-clients" in the Payara Server Application Development section.

Superelements

java-web-start-access (`glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`jsp-config`

Specifies JSP configuration information.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `jsp-config` element.

Table 87. `jsp-config` Subelements

Element	Required	Description
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Properties

The default property values are tuned for development of JSP files at the cost of performance. To maximize performance, set `jsp-config` properties to these non-default values:

- `development` - `false` (as an alternative, set to `true` and give `modificationTestInterval` a large value)
- `mappedfile` - `false`
- `trimSpaces` - `true`
- `suppressSmap` - `true`
- `fork` - `false` (on Solaris)
- `classdebuginfo` - `false`

The following table describes properties for the `jsp-config` element.

Table 88. `jsp-config` Properties

Property	Default	Description
`checkInterval`	`0`	If `development` is set to `false` and `checkInterval` is greater than zero, background compilations are enabled. The `checkInterval` is the time in seconds between checks to see if a JSP file needs to be recompiled.

Property	Default	Description
`classdebuginfo`	`true`	Specifies whether the generated Java servlets are compiled with the debug option set (`-g` for `javac`).
`classpath`	created dynamically based on the current web application	Specifies the classpath to use when compiling generated servlets.
`compiler`	`javac`	Specifies the compiler Ant uses to compile JSP files. See the Ant documentation for more information: https://ant.apache.org/manual/index.html
`compilerSourceVM`	Depends on Payara Server's Java runtime	Specifies the JDK release with which source compatibility of the generated servlets is provided. Same as the `-source` release option of `javac`.
`compilerTargetVM`	Depends on Payara Server's Java runtime	Specifies the Virtual Machine for the Java platform (JVM software) version for which the servlet class files are generated. Same as the `-target` release option of `javac`.
`defaultBufferNone`	`false`	If `true`, the default for the `buffer` attribute of the `page` directive is `none`.
`development`	`true`	If set to `true`, enables development mode, which allows JSP files to be checked for modification. Specify the frequency at which JSPs are checked using the `modificationTestInterval` property.
`dumpSmap`	`false`	If set to `true`, dumps SMAP information for JSR 45 debugging to a file. Set to `false` if `suppressSmap` is `true`.
`enablePooling`	`true`	If set to `true`, tag handler pooling is enabled.
`enableTldValidation`	`false`	If set to `true`, all Tag Library Descriptor (TLD) files referenced by the web application are validated against their underlying schema or DTD file.

Property	Default	Description
`errorOnUseBeanInvalidClassAttribute`	`false`	If set to `true`, issues an error when the value of the `class` attribute in a `useBean` action is not a valid bean class.
`fork`	`true`	Specifies that Ant forks the compiling of JSP files, using a JVM machine separate from the one in which Tomcat is running.
`genStrAsByteArray`	`true`	If `true`, text strings are generated as bytes (encoded with the page encoding), if the page is not buffered.
`genStrAsCharArray`	`false`	If set to `true`, generates text strings as `char` arrays, which improves performance in some cases.
`httpMethods`	`*` for all methods	Specifies a comma separated list of HTTP methods supported by the `JspServlet`.
`ieClassId`	`clsid:8AD9C840-044E-11D1-B3E9-00805F499D93`	Specifies the Java plug-in COM class ID for Internet Explorer. Used by the `<jsp:plugin>` tags.
`ignoreJspFragmentErrors`	`false`	If set to `true`, instructs the compiler to ignore any JSP precompilation errors pertaining to statically included JSP segments that, despite not being top level JSP files, use the `.jsp` or `.jspx` extension (instead of the recommended `.jspf`).
`initialCapacity`	`32`	Specifies the initial capacity of the `HashMap` that maps JSP files to their corresponding servlets.
`javaEncoding`	`UTF8`	Specifies the encoding for the generated Java servlet. This encoding is passed to the Java compiler that is used to compile the servlet as well. By default, the web container tries to use `UTF8`. If that fails, it tries to use the `javaEncoding` value. For encodings, see: `https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html`

Property	Default	Description
`keepgenerated`	`true` with JDK 5 and before and for `jspc`, otherwise `false`	If set to `true`, keeps the generated Java files. If `false`, deletes the Java files.
`mappedfile`	`true`	If set to `true`, generates static content with one print statement per input line, to ease debugging.
`modificationTestInterval`	`0`	Specifies the frequency in seconds at which JSPs are checked for modification. A value of `0` causes the JSP to be checked on every access. Used only if `development` is set to `true`.
`reload-interval`	`0`	Specifies the frequency in seconds at which JSP files are checked for modifications. Setting this value to `0` checks JSP files for modifications on every request. Setting this value to `-1` disables checks for JSP modifications and JSP recompilation.
`saveBytecode`	`true` for `jspc`, otherwise `false`	If `true`, generated byte code is saved to `.class` files. This option is meaningful only when the Java compiler API, JSR 199 (available with and used as the default on Java 6) is used for `javac` compilations.
`scratchdir`	The default work directory for the web application	Specifies the working directory created for storing all the generated code.
`suppressSmap`	`false`	If set to `true`, generation of SMAP information for JSR 45 debugging is suppressed.
`trimSpaces`	`false`	If set to `true`, trims white spaces in template text between actions or directives.

Property	Default	Description
`usePrecompiled`	`false`	If set to `true`, an accessed JSP file is not compiled. Its precompiled servlet class is used instead. It is assumed that JSP files have been precompiled, and their corresponding servlet classes have been bundled in the web application's `WEB-INF/lib` or `WEB-INF/classes` directory.

The default value of the compiler property for this element has changed from 1.5 to 1.8, denoting a change from JDK5 to JDK8.

You can change this to another value by editing the jsp-config element in the payara-web.xml file.

`keep-state`
If set to `true`, retains web sessions, stateful session bean instances, and persistently created EJB timers across redeployments. The `--keepstate` option of the "`redeploy`" subcommand takes precedence. The default for both is `false`.

Some changes to an application between redeployments prevent this feature from working properly. For example, do not change the set of instance variables in the SFSB bean class.

For web applications, this feature is applicable only if in the `payara-web.xml` file the `persistence-type` attribute of the session-manager element is `file`.

For stateful session bean instances, the persistence type without high availability is set in the server (the `sfsb-persistence-type` attribute) and must be set to `file`, which is the default and recommended value.

If any active web session, SFSB instance, or EJB timer fails to be preserved or restored, none of these will be available when the redeployment is complete. However, the redeployment continues and a warning is logged.

To preserve active state data, Payara Server serializes the data and saves it in memory.

To restore the data, the class loader of the newly redeployed application deserializes the data that was previously saved.

Superelements

payara-application (`glassfish-application.xml` (deprecated) and `payara-application.xml`), payara-web-app (`payara-web.xml`), payara-ejb-jar (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`key-field`

Specifies a component of the key used to look up and extract cache entries. The web container looks for the named parameter, or field, in the specified scope.

If this element is not present, the web container uses the Servlet Path (the path section that corresponds to the servlet mapping that activated the current request). See the Servlet 6.0 specification, section 3.6, for details on the Servlet Path.

Superelements

cache-mapping (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none

Attributes

The following table describes attributes for the `key-field` element.

Table 89. `key-field` Attributes

Attribute	Default	Description
`name`	none	Specifies the input parameter name.

Attribute	Default	Description
`scope`	`request.parameter`	(optional) Specifies the scope from which the input parameter is retrieved. Allowed values are `context.attribute`, `request.header`, `request.parameter`, `request.cookie`, `session.id`, and `session.attribute`.

`local-home-impl`

Specifies the fully-qualified class name of the generated `EJBLocalHome impl` class.

> **NOTE** This value is automatically generated by the server at deployment or redeployment time. Do not specify it or change it after deployment.

Superelements

gen-classes (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`local-impl`

Specifies the fully-qualified class name of the generated `EJBLocalObject impl` class.

> **NOTE** This value is automatically generated by the server at deployment or redeployment time. Do not specify it or change it after deployment.

Superelements

gen-classes (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`localpart`

Specifies the local part of a QNAME.

Superelements

service-qname,wsdl-port (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`log-service`

Specifies configuration settings for the log file.

Superelements

client-container (`sun-acc.xml`)

Subelements

The following table describes subelements for the `log-service` element.

Table 90. `log-service` subelement

Element	Required	Description
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `log-service` element.

Table 91. `log-service` attributes

Attribute	Default	Description
`log-file`	`your-ACC-dir/logs/client.log`	(optional) Specifies the file where the application client container logging information is stored.
`level`	`SEVERE`	(optional) Sets the base level of severity. Messages at or above this setting get logged to the log file.

`login-config`

Specifies the authentication configuration for an EJB web service endpoint. Not needed

for servlet web service endpoints. A servlet's security configuration is contained in the `web.xml` file.

Superelements

webservice-endpoint (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `login-config` element.

Table 92. `login-config` subelements

Element	Required	Description
auth-method	only one	Specifies the authentication method.
realm	zero or one	Specifies the name of the realm used to process all authentication requests.

`mail-resource`

Defines a Jakarta Mail (`jakarta.mail.Session`) resource.

Superelements

resources (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `mail-resource` element.

Table 93. `mail-resource` Subelements

Element	Required	Description
description	zero or one	Contains a text description of this element.
property-with-attributes	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the `mail-resource` element.

Table 94. `mail-resource` Attributes

Attribute	Default	Description
`jndi-name`	none	Specifies the JNDI name for the resource.
`store-protocol`	`imap`	(optional) Specifies the storage protocol service, which connects to a mail server, retrieves messages, and saves messages in folder(s). Allowed values are `imap`, `pop3`, `imaps`, and `pop3s`.
`store-protocol-class`	`org.eclipse.angus.mail.imap.IMAPStore`	(optional) Specifies the service provider implementation class for storage. Allowed values are: • `org.eclipse.angus.mail.imap.IMAPStore` • `org.eclipse.angus.mail.pop3.POP3Store` • `org.eclipse.angus.mail.imap.IMAPSSLStore` • `org.eclipse.angus.mail.pop3.POP3SSLStore`
`transport-protocol`	`smtp`	(optional) Specifies the transport protocol service, which sends messages. Allowed values are `smtp` and `smtps`.

`transport-protocol-class`	`org.eclipse.angus.mail.smtp.SMTPTransport`	(optional) Specifies the service provider implementation class for transport. Allowed values are:
		`org.eclipse.angus.mail.smtp.SMTPTransport`
		`org.eclipse.angus.mail.smtp.SMTPSSLTransport`

`host`	none	The mail server host name.
`user`	none	The mail server username.
`from`	none	The email address the mail server uses to indicate the message sender.
`debug`	`false`	(optional) Determines whether debugging for this resource is enabled.

Attribute	Default	Description
`object-type`	`user`	(optional) Defines the type of the resource. Allowed values are: • `system-all` - A system resource for all server instances and the domain application server. • `system-admin` - A system resource only for the domain application server. • `system-instance` - A system resource for all server instances only. • `user` - A user resource.
`enabled`	`true`	(optional) Determines whether this resource is enabled at runtime.

Properties

You can set properties for the `mail-resource` element and then get these properties in a Jakarta Mail `Session` object later. Every property name must start with a `mail-` prefix.

The Payara Server changes the dash (`-`) character to a period (`.`) in the name of the property, then saves the property to the `MailConfiguration` and Jakarta Mail `Session` objects. If the name of the property doesn't start with `mail-`, the property is ignored.

For example, to define the property `mail.password` in a Jakarta Mail Session object, first edit `payara-resources.xml` as follows:

```
<mail-resource jndi-name="mail/Session">
    <property name="mail-password" value="adminadmin"/>
    <property name="mail-password" value="adminadmin"/>
</mail-resource>
```

IMPORTANT	glassfish-resources.xml is deprecated and support will be removed in a future release. It is recommended to use payara-

> resources.xml instead.

After getting the Jakarta Mail `Session` object, get the `mail.password` property to retrieve the value `adminadmin`, as follows:

```
public class Component{

    public void businessMethod(){
        var password = session.getProperty("mail.password");
    }
}
```

For more information about Jakarta Mail properties, see Jakarta Mail API Documentation (`https://jakarta.ee/specifications/mail`).

`manager-properties`

Specifies session manager properties.

Superelements

session-manager (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `manager-properties` element.

Table 95. `manager-properties` Subelements

Element	Required	Description
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Properties

The following table describes properties for the `manager-properties` element.

Table 96. `manager-properties` Properties

Property	Default	Description
`reapIntervalSeconds`	`60`	Specifies the number of seconds between checks for expired sessions. This is also the interval at which sessions are passivated if `maxSessions` is exceeded.

Property	Default	Description
`maxSessions`	`-1`	Specifies the maximum number of sessions that are permitted in the cache, or `-1` for no limit. After this, an attempt to create a new session causes an `IllegalStateException` to be thrown. If the `persistence-type` attribute of the parent session-manager element is `file`, the session manager passivates sessions to the persistent store when this maximum is reached.

`sessionFilename`	empty string	Specifies the absolute or relative path to the directory in which the session state is preserved between application restarts, if preserving the state is possible. A relative path is relative to the temporary directory for this web module, one of the following:
		
		`domain-dir/generated/jsp/module-name`
		
		`domain-dir/generated/jsp/app-name/module-name`
		
		By default, this property's value is set to an empty string, which disables this property and does not preserve the session state.
		
		Applicable only if the `persistence-type` attribute of the parent session-manager element is `memory`.

Property	Default	Description
`persistenceFrequency`	`web-method`	Specifies how often the session state is stored. Allowed values are as follows: • `web-method` - The session state is stored at the end of each web request prior to sending a response back to the client. This mode provides the best guarantee that the session state is fully updated in case of failure. • `time-based` - The session state is stored in the background at the frequency set by `reapIntervalSeconds`. This mode provides less of a guarantee that the session state is fully updated. However, it can provide a significant performance improvement because the state is not stored after each request. Applicable only if the `persistence-type` attribute of the parent session-manager element is `hazelcast`.

`max-cache-size`

Specifies the maximum number of beans allowable in cache. A value of zero indicates an unbounded cache. In reality, there is no hard limit. The max-cache-size limit is just a hint to the cache implementation. Default is 512.

Applies to stateful session beans and entity beans.

Superelements

bean-cache (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`max-pool-size`

Specifies the maximum number of bean instances in the pool. Values are from 0 (1 for message-driven bean) to MAX_INTEGER. A value of 0 means the pool is unbounded.

Default is 64.

Applies to all beans.

Superelements

bean-pool (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

Subelements

`mdb-connection-factory`

Specifies the connection factory associated with a message-driven bean. Queue or Topic type must be consistent with the Jakarta Messaging Destination type associated with the message-driven bean class.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `mdb-connection-factory` element.

Table 97. `mdb-connection-factory` Subelements

Element	Required	Description
jndi-name	only one	Specifies the absolute `jndi-name`.
default-resource-principal	zero or one	Specifies the default sign-on (name/password) to the resource manager.

`mdb-resource-adapter`

Specifies runtime configuration information for a message-driven bean.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `mdb-resource-adapter` element.

Table 98. `mdb-resource-adapter` subelements

Element	Required	Description
resource-adapter-mid	zero or one	Specifies a resource adapter module ID.
activation-config	one or more	Specifies an activation configuration.

`message`

Specifies the methods or operations to which message security requirements apply.

Superelements

message-security (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

The following table describes subelements for the `message` element.

Table 99. `message` Subelements

Element	Required	Description
java-method	zero or one	Specifies the methods or operations to which message security requirements apply.
operation-name	zero or one	Specifies the WSDL name of an operation of a web service.

`message-destination`

Specifies the name of a logical `message-destination` defined within an application. The `message-destination-name` matches the corresponding `message-destination-name` in the corresponding Jakarta EE deployment descriptor file. Use when the message destination reference in the corresponding Jakarta EE deployment descriptor file specifies a `message-destination-link` to a logical `message-`

`destination`.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`), enterprise-beans (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`), payara-application-client (`glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

The following table describes subelements for the `message-destination` element.

Table 100. `message-destination` subelements

Element	Required	Description
message-destination-name	only one	Specifies the name of a logical message destination defined within the corresponding Jakarta EE deployment descriptor file.
jndi-name	only one	Specifies the `jndi-name` of the associated entity.

`message-destination-name`

Specifies the name of a logical message destination defined within the corresponding Jakarta EE deployment descriptor file.

Superelements

message-destination (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`message-destination-ref`

Directly binds a message destination reference to the JNDI name of a `Queue`, `Topic`, or other physical destination. Use only when the message destination reference in the

corresponding Jakarta EE deployment descriptor file does not specify a `message-destination-link` to a logical `message-destination`.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`), ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`), payara-application-client (`glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

The following table describes subelements for the `message-destination-ref` element.

Table 101. `message-destination-ref` subelements

Element	Required	Description
message-destination-ref-name	only one	Specifies the name of a physical message destination defined within the corresponding Jakarta EE deployment descriptor file.
jndi-name	only one	Specifies the `jndi-name` of the associated entity.

`message-destination-ref-name`

Specifies the name of a physical message destination defined within the corresponding Jakarta EE deployment descriptor file.

Superelements

message-destination-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`message-security`

Specifies message security requirements.

- If the grandparent element is webservice-endpoint, these requirements pertain to request and response messages of the endpoint.
- If the grandparent element is port-info, these requirements pertain to the port of the referenced service.

Superelements

message-security-binding (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

The following table describes subelements for the `message-security` element.

Table 102. `message-security` Subelements

Element	Required	Description
message	one or more	Specifies the methods or operations to which message security requirements apply.
request-protection	zero or one	Defines the authentication policy requirements of the application's request processing.
response-protection	zero or one	Defines the authentication policy requirements of the application's response processing.

`message-security-binding`

Specifies a custom authentication provider binding for a parent webservice-endpoint or port-info element in one or both of these ways:

- By binding to a specific provider
- By specifying the message security requirements enforced by the provider

Superelements

webservice-endpoint, port-info (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

The following table describes subelements for the `message-security-binding` element.

Table 103. `message-security-binding` Subelements

Element	Required	Description
message-security	zero or more	Specifies message security requirements.

Attributes

The following table describes attributes for the `message-security-binding` element.

Table 104. `message-security-binding` Attributes

Attribute	Default	Description
`auth-layer`	none	Specifies the message layer at which authentication is performed. The value must be `SOAP`.
`provider-id`	none	(optional) Specifies the authentication provider used to satisfy application-specific message security requirements. If this attribute is not specified, a default provider is used, if it is defined for the message layer. If no default provider is defined, authentication requirements defined in the `message-security-binding` are not enforced.

`message-security-config`

Specifies configurations for message security providers.

Superelements

client-container (`sun-acc.xml`)

Subelements

The following table describes subelements for the `message-security-config` element.

Table 105. `message-security-config` Subelements

Element	Required	Description
provider-config	one or more	Specifies a configuration for one message security provider.

Attributes

The following table describes attributes for the `message-security-config` element.

Table 106. `message-security-config` Attributes

Attribute	Default	Description
`auth-layer`	none	Specifies the message layer at which authentication is performed. The value must be `SOAP`.
`default-provider`	none	(optional) Specifies the server provider that is invoked for any application not bound to a specific server provider.
`default-client-provider`	none	(optional) Specifies the client provider that is invoked for any application not bound to a specific client provider.

`method`

Specifies a bean method.

Superelements

checkpoint-at-end-of-method, flush-at-end-of-method (`glassfish-ejb-jar.xml`

(deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `method` element.

Table 107. `method` Subelements

Element	Required	Description
description	zero or one	Specifies an optional text description.
ejb-name	zero or one	Matches the `ejb-name` in the corresponding `ejb-jar.xml` file.
method-name	only one	Specifies a method name.
method-intf	zero or one	Specifies the method interface to distinguish between methods with the same name in different interfaces.
method-params	zero or one	Specifies fully qualified Java type names of method parameters.

`method-intf`

Specifies the method interface to distinguish between methods with the same name in different interfaces. Allowed values are `Home`, `Remote`, `LocalHome`, and `Local`.

Superelements

method (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`method-name`

Specifies a method name or `*` (an asterisk) for all methods. If a method is overloaded, specifies all methods with the same name.

Superelements

java-method (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-`

`client.xml`); method (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

Examples

```
<method-name>findTeammates</method-name>
```

```
<method-name>*</method-name>
```

`method-param`

Specifies the fully qualified Java type name of a method parameter.

Superelements

method-params (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`method-params`

Specifies fully qualified Java type names of method parameters.

Superelements

java-method (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`); method (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `method-params` element.

Table 108. `method-params` Subelements

Element	Required	Description
method-param	zero or more	Specifies the fully qualified Java type name of a method parameter.

`name`

Specifies the name of the entity.

Superelements

call-property, default-resource-principal, stub-property (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`); enterprise-beans, principal, property-with-subelements (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`namespaceURI`

Specifies the namespace URI.

Superelements

service-qname, wsdl-port (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`operation-name`

Specifies the WSDL name of an operation of a web service.

Superelements

message (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-`

`ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`parameter-encoding`

Specifies the default request character encoding and how the web container decodes parameters from forms according to a hidden field value.

If both the payara-web-app and locale-charset-info elements have `parameter-encoding` subelements, the subelement of `payara-web-app` takes precedence. For encodings, see `https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html`.

Superelements

locale-charset-info, payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none

Attributes

The following table describes attributes for the `parameter-encoding` element.

Table 109. `parameter-encoding` Attributes

Attribute	Default	Description
`form-hint-field`	none	(optional) The name of the hidden field in the form. This field specifies the character encoding the web container uses for `request.getParameter` and `request.getReader` calls when the charset is not set in the request's `content-type` header.

Attribute	Default	Description
`default-charset`	`ISO-8859-1`	(optional) The default request character encoding.

`pass-by-reference`

Specifies the passing method used by a servlet or enterprise bean calling a remote interface method in another bean that is colocated within the same process.

- If `false` (the default if this element is not present), this application uses pass-by-value semantics.
- If `true`, this application uses pass-by-reference semantics.

> **NOTE** The `pass-by-reference` element only applies to remote calls. As defined in the Jakarta Enterprise Beans 4.0 specification, section 3.2.3, calls to local interfaces use pass-by-reference semantics.

If the `pass-by-reference` element is set to its default value of `false`, the passing semantics for calls to remote interfaces comply with the Jakarta Enterprise Beans 4.0 specification, section 3.2.3. If set to `true`, remote calls involve pass-by-reference semantics instead of pass-by-value semantics, contrary to this specification.

Portable programs cannot assume that a copy of the object is made during such a call, and thus that it's safe to modify the original. Nor can they assume that a copy is not made, and thus that changes to the object are visible to both caller and callee. When this element is set to `true`, parameters and return values should be considered read-only. The behavior of a program that modifies such parameters or return values is undefined.

When a servlet or enterprise bean calls a remote interface method in another bean that is colocated within the same process, by default Payara Server makes copies of all the call parameters in order to preserve the pass-by-value semantics. This increases the call overhead and decreases performance.

However, if the calling method does not change the object being passed as a parameter,

it is safe to pass the object itself without making a copy of it. To do this, set the pass-by-reference value to `true`.

The setting of this element in the `payara-application.xml` file applies to all EJB modules in the application. For an individually deployed EJB module, you can set the same element in the `payara-ejb-jar.xml` file. If `pass-by-reference` is used at both the bean and application level, the bean level takes precedence.

Superelements

payara-application (`payara-application.xml`), ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`password`

Specifies the password for the principal.

Superelements

default-resource-principal (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`per-request-load-balancing`

Specifies the per-request load balancing behavior of EJB 2.x and 3.x remote client invocations on a stateless session bean. If set to `true`, per-request load balancing is enabled for the associated stateless session bean. If set to `false` or not set, per-request load balancing is not enabled. The default is `false`.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`pool-idle-timeout-in-seconds`

Specifies the maximum time, in seconds, that a bean instance is allowed to remain idle in the pool. When this timeout expires, the bean instance in a pool becomes a candidate for passivation or deletion. This is a hint to the server. A value of 0 specifies that idle beans remain in the pool indefinitely. Default value is 600.

Applies to stateless session beans, entity beans, and message-driven beans.

NOTE	For a stateless session bean or a message-driven bean, the bean is removed (garbage collected) when the timeout expires.

Superelements

bean-pool (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`port-component-name`

Specifies a unique name for a port component within a web or EJB module.

Superelements

webservice-endpoint (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`port-info`

Specifies information for a port within a web service reference.

Either a `service-endpoint-interface` or a `wsdl-port` or both must be specified. If both are specified, `wsdl-port` specifies the port that the container chooses for container-managed port selection.

The same `wsdl-port` value must not appear in more than one `port-info` element within the same `service-ref`.

If a `service-endpoint-interface` is using container-managed port selection, its value must not appear in more than one `port-info` element within the same `service-ref`.

Superelements

service-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

The following table describes subelements for the `port-info` element.

Table 110. `port-info` subelements

Element	Required	Description
service-endpoint-interface	zero or one	Specifies the web service reference name relative to `java:comp/env`.
wsdl-port	zero or one	Specifies the WSDL port.
stub-property	zero or more	Specifies JAX-RPC property values that are set on a `javax.xml.rpc.Stub` object before it is returned to the web service client.
call-property	zero or more	Specifies JAX-RPC property values that are set on a `javax.xml.rpc.Call` object before it is returned to the web service client.
message-security-binding	zero or one	Specifies a custom authentication provider binding.

`principal`

Defines a user name on the platform.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`); security-map (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `principal` element.

Table 111. `principal` Subelements

Element	Required	Description
name	only one	Specifies the name of the user.

`principal-map`

Maps an EIS principal to a principal defined in the Payara Server domain.

Superelements

work-security-map (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

none

Attributes

The following table describes attributes for the `principal-map` element.

Table 112. `principal-map` Attributes

Attribute	Default	Description
`eis-principal`	none	Specifies an EIS principal.
`mapped-principal`	none	Specifies a principal defined in the Payara Server domain.

`principal-name`

Contains the principal username.

In an enterprise bean, specifies the principal username that has the `run-as` role

specified.

Superelements

security-role-mapping (`glassfish-application.xml` (deprecated) and `payara-application.xml`, `glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`), servlet (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none - contains data

Attributes

The following table describes attributes for the `principal-name` element.

Table 113. `principal-name` Attributes

Attribute	Default	Description
`class-name`	`com.sun.enterprise.deployment.PrincipalImpl`	(optional) Specifies the custom principal implementation class corresponding to the named principal.

`property` (with Attributes)

Specifies the name and value of a property. A property adds configuration information to its parent element that is one or both of the following:

- Optional with respect to Payara Server
- Needed by a system or object that Payara Server doesn't have knowledge of, such as an LDAP server or a Java class

Superelements

cache, cache-helper, class-loader, cookie-properties, default-helper, manager-properties, session-properties, store-properties, payara-web-app, valve, webservice-endpoint (`glassfish-web.xml` (deprecated) and `payara-web.xml`); auth-realm, client-container, client-credential, log-service, provider-config (`sun-acc.xml`); admin-object-

resource, connector-connection-pool, connector-resource, custom-resource, external-jndi-resource, jdbc-connection-pool, "`jdbc-resource`", mail-resource, resource-adapter-config (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`), payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `property` element.

Table 114. `property` Subelements

Element	Required	Description
description	zero or one	Specifies an optional text description of a property.

NOTE The `property` element in the `sun-acc.xml` file has no subelements.

Attributes

The following table describes attributes for the `property` element.

Table 115. `property` Attributes

Attribute	Default	Description
`name`	none	Specifies the name of the property.
`value`	none	Specifies the value of the property.

Example

```
<property name="reapIntervalSeconds" value="20" />
```

`property` (with Subelements)

Specifies the name and value of a property. A property adds configuration information to its parent element that is one or both of the following:

- Optional with respect to Payara Server
- Needed by a system or object that Payara Server doesn't have knowledge of, such as

an LDAP server or a Jakarta class

Superelements

enterprise-beans, webservice-endpoint (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `property` element.

Table 116. `property` subelements

Element	Required	Description
name	only one	Specifies the name of the property.
value	only one	Specifies the value of the property.

Example

```
<property>
   <name>use-unique-table-names</name>
   <value>true</value>
</property>
```

`provider-config`

Specifies a configuration for one message security provider.

Although the `request-policy` and `response-policy` subelements are optional, the `provider-config` element does nothing if they are not specified.

Use property subelements to configure provider-specific properties. Property values are passed to the provider when its `initialize` method is called.

Superelements

message-security-config (`sun-acc.xml`)

Subelements

The following table describes subelements for the `provider-config` element.

Table 117. `provider-config` Subelements

Element	Required	Description
request-policy	zero or one	Defines the authentication policy requirements of the authentication provider's request processing.
response-policy	zero or one	Defines the authentication policy requirements of the authentication provider's response processing.
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `provider-config` element.

Table 118. `provider-config` Attributes

Attribute	Default	Description
`provider-id`	none	Specifies the provider ID.
`provider-type`	none	Specifies whether the provider is a `client`, `server`, or `client-server` authentication provider.
`class-name`	none	Specifies the Jakarta implementation class of the provider. Client authentication providers must implement the com.sun.enterprise.security.jauth.ClientAuthModule interface. Server authentication providers must implement the com.sun.enterprise.security.jauth.ServerAuthModule interface. Client-server providers must implement both interfaces.

`realm`

Specifies the name of the realm used to process all authentication requests associated with this application. If this element is not specified or does not match the name of a configured realm, the default realm is used. For more information about realms, see "realm-configuration" in the Payara Server Application Development section.

Superelements

payara-application (`glassfish-application.xml` (deprecated) and `payara-`

`application.xml`), as-context, login-config (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`refresh-field`

Specifies a field that gives the application component a programmatic way to refresh a cached entry.

Superelements

cache-mapping (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none

Attributes

The following table describes attributes for the `refresh-field` element.

Table 119. `refresh-field` Attributes

Attribute	Default	Description
`name`	none	Specifies the input parameter name.
`scope`	`request.parameter`	(optional) Specifies the scope from which the input parameter is retrieved. Allowed values are `context.attribute`, `request.header`, `request.parameter`, `request.cookie`, `session.id`, and `session.attribute`.

`refresh-period-in-seconds`

Specifies the rate at which a read-only-bean must be refreshed from the data source. If the value is less than or equal to zero, the bean is never refreshed; if the value is greater than zero, the bean instances are refreshed at the specified interval. This rate is just a hint to the container. Default is 0 (no refresh).

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`removal-timeout-in-seconds`

Specifies the amount of time a bean instance can remain idle in the container before it is removed (timeout). A value of 0 specifies that the container does not remove inactive beans automatically. The default value is 5400.

If `removal-timeout-in-seconds` is less than or equal to `cache-idle-timeout-in-seconds`, beans are removed immediately without being passivated.

Applies to stateful session beans.

For related information, see cache-idle-timeout-in-seconds.

Superelements

bean-cache (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`remote-home-impl`

Specifies the fully-qualified class name of the generated `EJBHome impl` class.

NOTE	This value is automatically generated by the server at deployment or redeployment time. Do not specify it or change it after deployment.

Superelements

gen-classes (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`remote-impl`

Specifies the fully-qualified class name of the generated `EJBObject impl` class.

NOTE	This value is automatically generated by the server at deployment or redeployment time. Do not specify it or change it after deployment.

Superelements

gen-classes (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`request-policy`

Defines the authentication policy requirements of the authentication provider's request processing.

Superelements

provider-config (`sun-acc.xml`)

Subelements

none

Attributes

The following table describes attributes for the `request-policy` element.

Table 120. `request-policy` Attributes

Attribute	Default	Description
`auth-source`	none	Specifies the type of required authentication, either `sender` (user name and password) or `content` (digital signature).
`auth-recipient`	none	Specifies whether recipient authentication occurs before or after content authentication. Allowed values are `before-content` and `after-content`.

`request-protection`

Defines the authentication policy requirements of the application's request processing.

Superelements

message-security (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none

Attributes

The following table describes attributes for the `request-protection` element.

Table 121. `request-protection` Attributes

Attribute	Default	Description
`auth-source`	none	Specifies the type of required authentication, either `sender` (user name and password) or `content` (digital signature).
`auth-recipient`	none	Specifies whether recipient authentication occurs before or after content authentication. Allowed values are `before-content` and `after-content`.

`required`

Specifies whether the authentication method specified in the auth-method element must be used for client authentication. The value is `true` or `false` (the default).

Superelements

as-context (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`res-ref-name`

Specifies the `res-ref-name` in the corresponding Jakarta EE deployment descriptor file `resource-ref` entry. The `res-ref-name` element specifies the name of a resource manager connection factory reference. The name must be unique within an enterprise bean.

Superelements

resource-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`resize-quantity`

Specifies the number of bean instances to be:

- Created, if a request arrives when the pool has less than steady-pool-size quantity of beans (applies to pools only for creation). If the pool has more than `steady-pool-size` minus resize-quantity of beans, then `resize-quantity` is still created.
- Removed, when the pool-idle-timeout-in-seconds timer expires and a cleaner thread removes any unused instances.
    - For caches, when max-cache-size is reached, `resize-quantity` beans are selected for passivation using the victim-selection-policy. In addition, the cache-idle-timeout-in-seconds or removal-timeout-in-seconds timers passivate beans from the cache.
    - For pools, when the max-pool-size is reached, `resize-quantity` beans are selected for removal. In addition, the pool-idle-timeout-in-seconds timer removes beans until `steady-pool-size` is reached.

Values are from 0 to MAX_INTEGER. The pool is not resized below the `steady-pool-size`. Default is 16.

Applies to stateless session beans, entity beans, and message-driven beans.

For EJB pools, the value can be defined in the EJB container. Default is 16.

For EJB caches, the value can be defined in the EJB container. Default is 32.

For message-driven beans, the value can be defined in the EJB container. Default is 2.

Superelements

bean-cache, bean-pool (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`resource-adapter-config`

Defines a connector (resource adapter) configuration. Stores configuration information for the resource adapter JavaBean in `property` subelements.

Superelements

resources (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `resource-adapter-config` element.

Table 122. `resource-adapter-config` *Subelements*

Element	Required	Description
property-with-attributes	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the `resource-adapter-config` element.

Table 123. `resource-adapter-config` *Attributes*

Attribute	Default	Description
`name`	none	(optional) Not used. See `resource-adapter-name`.
`thread-pool-ids`	none	(optional) Specifies a comma-separated list of the names of thread pools.
`object-type`	`user`	(optional) Defines the type of the resource. Allowed values are: • `system-all` - A system resource for all server instances and the domain application server. • `system-admin` - A system resource only for the domain application server. • `system-instance` - A system resource for all server instances only. • `user` - A user resource.
`resource-adapter-name`	none	Specifies the name of a deployed connector module or application. If the resource adapter is embedded in an application, then it is `app_name#rar_name`.

Properties

Properties of the `resource-adapter-config` element are the names of setter methods of the `resourceadapter-class` element in the `ra.xml` file, which defines the class name of the resource adapter JavaBean. Any properties defined here override the default values present in `ra.xml`.

`resource-adapter-mid`

Specifies the module ID of the resource adapter that is responsible for delivering messages to the message-driven bean.

Superelements

mdb-resource-adapter (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`resource-env-ref`

Maps the `res-ref-name` in the corresponding Jakarta EE deployment descriptor file `resource-env-ref` entry to the absolute `jndi-name` of a resource.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`), ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`), payara-application-client (`glassfish-application-client.xml`)

Subelements

The following table describes subelements for the `resource-env-ref` element.

Table 124. `resource-env-ref` Subelements

Element	Required	Description
resource-env-ref-name	only one	Specifies the `res-ref-name` in the corresponding Jakarta EE deployment descriptor file `resource-env-ref` entry.
jndi-name	only one	Specifies the absolute `jndi-name` of a resource.

Example

```
<resource-env-ref>
   <resource-env-ref-name>jms/StockQueueName</resource-env-ref-name>
   <jndi-name>jms/StockQueue</jndi-name>
</resource-env-ref>
```

`resource-env-ref-name`

Specifies the `res-ref-name` in the corresponding Jakarta EE deployment descriptor file `resource-env-ref` entry.

Superelements

resource-env-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-`

`application-client.xml`)

Subelements

none - contains data

`resource-ref`

Maps the `res-ref-name` in the corresponding Jakarta EE deployment descriptor file `resource-ref` entry to the absolute `jndi-name` of a resource.

> **NOTE** Connections acquired from JMS connection factories are not shareable in the current release of Payara Server. The `res-sharing-scope` element in the `ejb-jar.xml` file `resource-ref` element is ignored for JMS connection factories.

When `resource-ref` specifies a JMS connection factory for the Open Message Queue, the `default-resource-principal` (name/password) must exist in the Message Queue user repository.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`), ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`), payara-application-client (`glassfish-application-client.xml`)

Subelements

The following table describes subelements for the `resource-ref` element.

Table 125. `resource-ref` Subelements

Element	Required	Description
res-ref-name	only one	Specifies the `res-ref-name` in the corresponding Jakarta EE deployment descriptor file `resource-ref` entry.
jndi-name	only one	Specifies the absolute `jndi-name` of a resource.
default-resource-principal	zero or one	Specifies the default principal (user) for the resource.

Example

```
<resource-ref>
   <res-ref-name>jdbc/EmployeeDBName</res-ref-name>
   <jndi-name>jdbc/EmployeeDB</jndi-name>
</resource-ref>
```

`resources`

Defines application-scoped resources for an enterprise application, web module, EJB module, connector module, or application client module. This is the root element; there can only be one `resources` element in a `glassfish-resources.xml` or `payara-resources.xml` file.

IMPORTANT	glassfish-resources.xml is deprecated and support will be removed in a future release. It is recommended to use payara-resources.xml instead.

See the-payara-resources.xml-file.

NOTE	You must specify a Java Naming and Directory Interface (JNDI) name for each resource. To avoid collisions with names of other enterprise resources in JNDI, and to avoid portability problems, all names in a Payara Server application should begin with the string `java:app/`.

Superelements

none

Subelements

The following table describes subelements for the `resources` element.

Table 126. `resources` Subelements

Element	Required	Description
custom-resource	zero or more	Defines a custom resource.
external-jndi-resource	zero or more	Defines a resource that resides in an external JNDI repository.

Element	Required	Description
"`jdbc-resource`"	zero or more	Defines a JDBC (Java Database Connectivity) resource.
mail-resource	zero or more	Defines a Jakarta Mail resource.
admin-object-resource	zero or more	Defines an administered object for an inbound resource adapter.
connector-resource	zero or more	Defines a connector (resource adapter) resource.
resource-adapter-config	zero or more	Defines a resource adapter configuration.
jdbc-connection-pool	zero or more	Defines the properties that are required for creating a JDBC connection pool.
connector-connection-pool	zero or more	Defines the properties that are required for creating a connector connection pool.
work-security-map	zero or more	Defines a work security map.

NOTE | Subelements of a `resources` element can occur in any order.

`response-policy`

Defines the authentication policy requirements of the authentication provider's response processing.

Superelements

provider-config (`sun-acc.xml`)

Subelements

none

Attributes

The following table describes attributes for the `response-policy` element.

Table 127. `response-policy` Attributes

Attribute	Default	Description
`auth-source`	none	Specifies the type of required authentication, either `sender` (user name and password) or `content` (digital signature).
`auth-recipient`	none	Specifies whether recipient authentication occurs before or after content authentication. Allowed values are `before-content` and `after-content`.

`response-protection`

Defines the authentication policy requirements of the application's response processing.

Superelements

message-security (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none

Attributes

The following table describes attributes for the `response-protection` element.

Table 128. `response-protection` Attributes

Attribute	Default	Description
`auth-source`	none	Specifies the type of required authentication, either `sender` (user name and password) or `content` (digital signature).
`auth-recipient`	none	Specifies whether recipient authentication occurs before or after content authentication. Allowed values are `before-content` and `after-content`.

`role-name`

Contains the `role-name` in the `security-role` element of the corresponding Jakarta EE deployment descriptor file.

Superelements

security-role-mapping (`glassfish-application.xml` (deprecated) and `payara-application.xml`` , `glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`sas-context`

Describes the sas-context fields.

Superelements

ior-security-config (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `sas-context` element.

Table 129. `sas-context` Subelements

Element	Required	Description
caller-propagation	only one	Specifies whether the target accepts propagated caller identities. The values are NONE, SUPPORTED, or REQUIRED.

`scanning-exclude` and `scanning-include`

Modern WAR and EAR files very often include a number of 3rd party JARs. In situations where some JARs require CDI scanning and others may break if scanned, these can now be explicitly included or excluded from such component scanning.

Both the `payara-application.xml` and the `payara-web.xml` files support the following directives:

```
<webapp>
    <scanning-exclude>*</scanning-exclude>
    <scanning-include>ejb*</scanning-include>
    <scanning-include>conflicting-web-library</scanning-include>
</webapp>
```

In the above example, all JARs will be excluded by default, then all JARs beginning with `ejb` will be scanned along with the JAR named `conflicting-web-library`.

Superelements

payara-application (`glassfish-application.xml` (deprecated) and `payara-application.xml`), payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none

`security`

Defines the SSL security configuration for IIOP/SSL communication with the target server.

Superelements

target-server (`sun-acc.xml`)

Subelements

The following table describes subelements for the `security` element.

Table 130. `security` Subelements

Element	Required	Description
ssl	only one	Specifies the SSL processing parameters.
cert-db	only one	Not implemented. Included for backward compatibility only.

`security-map`

Maps the principal received during servlet or EJB authentication to the credentials accepted by the EIS. This mapping is optional. It is possible to map multiple Payara

Server principals to the same back-end principal.

This is different from a work-security-map, which maps a principal associated with an incoming work instance to a principal in the Payara Server's security domain.

Superelements

connector-connection-pool (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `security-map` element.

Table 131. `security-map` Subelements

Element	Required	Description
principal	one or more	Contains the principal of the servlet or EJB client.
user-group	one or more	Contains the group to which the principal belongs.
backend-principal	only one	Specifies the username and password required by the EIS.

Attributes

The following table describes attributes for the `security-map` element.

Table 132. `security-map` Attributes

Attribute	Default	Description
`name`	none	Specifies a name for the security mapping.

`security-role-mapping`

Maps roles to users or groups in the currently active realm. See "realm-configuration" in the Payara Server Application Development section.

The role mapping element maps a role, as specified in the EJB JAR `role-name` entries, to a environment-specific user or group. If it maps to a user, it must be a concrete user which exists in the current realm, who can log into the server using the current

authentication method. If it maps to a group, the realm must support groups and the group must be a concrete group which exists in the current realm. To be useful, there must be at least one user in that realm who belongs to that group.

Superelements

payara-application (`glassfish-application.xml` (deprecated) and `payara-application.xml``), payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`), payara-ejb-jar (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `security-role-mapping` element.

Table 133. `security-role-mapping` Subelements

Element	Required	Description
role-name	only one	Contains the `role-name` in the `security-role` element of the corresponding Jakarta EE deployment descriptor file.
principal-name	one or more if no `group-name`, otherwise zero or more	Contains a principal (user) name in the current realm. In an enterprise bean, the principal must have the run-as role specified.
group-name	one or more if no `principal-name`, otherwise zero or more	Contains a group name in the current realm.

`service-endpoint-interface`

Specifies the web service reference name relative to `java:comp/env`.

Superelements

port-info (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-`

`application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`service-impl-class`

Specifies the name of the generated service implementation class.

Superelements

service-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`service-qname`

Specifies the WSDL service element that is being referred to.

Superelements

service-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`); webservice-endpoint (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `service-qname` element.

Table 134. `service-qname` subelements

Element	Required	Description
namespaceuri	only one	Specifies the namespace URI.
localpart	only one	Specifies the local part of a QNAME.

`service-ref`

Specifies runtime settings for a web service reference. Runtime information is only needed in the following cases:

- To define the port used to resolve a container-managed port
- To define the default Stub/Call property settings for Stub objects
- To define the URL of a final WSDL document to be used instead of the one associated with the `service-ref` in the standard Jakarta EE deployment descriptor

Superelements

payara-web-app (<code>glassfish-web.xml</code> (deprecated) and <code>payara-web.xml</code>), ejb (<code>glassfish-ejb-jar.xml</code> (deprecated) and <code>payara-ejb-jar.xml</code>), payara-application-client (<code>glassfish-application-client.xml</code> (deprecated) and `payara-application-client.xml</code>)

Subelements

The following table describes subelements for the `service-ref` element.

Table 135. `service-ref` subelements

Element	Required	Description
service-ref-name	only one	Specifies the web service reference name relative to `java:comp/env`.
port-info	zero or more	Specifies information for a port within a web service reference.
call-property	zero or more	Specifies JAX-RPC property values that can be set on a `javax.xml.rpc.Call` object before it is returned to the web service client.

Element	Required	Description
wsdl-override	zero or one	Specifies a valid URL pointing to a final WSDL document.
service-impl-class	zero or one	Specifies the name of the generated service implementation class.
service-qname	zero or one	Specifies the WSDL service element that is being referenced.

`service-ref-name`

Specifies the web service reference name relative to `java:comp/env`.

Superelements

service-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml``)

Subelements

none - contains data

`servlet`

Specifies a principal name for a servlet. Used for the `run-as` role defined in `web.xml`.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `servlet` element.

Table 136. `servlet` Subelements

Element	Required	Description
servlet-name	only one	Contains the name of a servlet, which is matched to a `servlet-name` in `web.xml`.

Element	Required	Description
principal-name	zero or one	Contains a principal (user) name in the current realm.
webservice-endpoint	zero or more	Specifies information about a web service endpoint.

`servlet-impl-class`

Specifies the automatically generated name of the servlet implementation class.

Superelements

webservice-endpoint (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`servlet-name`

Specifies the name of a servlet, which is matched to a `servlet-name` in `web.xml`. This name must be present in `web.xml`.

Superelements

cache-mapping, servlet (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none - contains data

`session-config`

Specifies session configuration information. Overrides the web container settings for an individual web module.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `session-config` element.

Table 137. `session-config` Subelements

Element	Required	Description
session-manager	zero or one	Specifies session manager configuration information.
session-properties	zero or one	Specifies session properties.
cookie-properties	zero or one	Specifies session cookie properties.

`session-manager`

Specifies session manager information.

Superelements

session-config (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `session-manager` element.

Table 138. `session-manager` Subelements

Element	Required	Description
manager-properties	zero or one	Specifies session manager properties.
store-properties	zero or one	Specifies session persistence (storage) properties.

Attributes

The following table describes attributes for the `session-manager` element.

Table 139. `session-manager` Attributes

Attribute	Default	Description
`persistence-type`	`memory`	(optional) Specifies the session persistence mechanism. Allowed values are `memory`, `file`, and `hazelcast`.

`session-properties`

Specifies session properties.

Superelements

session-config (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `session-properties` element.

Table 140. `session-properties` Subelements

Element	Required	Description
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Properties

The following table describes properties for the `session-properties` element.

Table 141. `session-properties` Properties

Property	Default	Description
`timeoutSeconds`	`1800`	Specifies the default maximum inactive interval (in seconds) for all sessions created in this web module. If set to `0` or less, sessions in this web module never expire. If a `session-timeout` element is specified in the `web.xml` file, the `session-timeout` value overrides any `timeoutSeconds` value. If neither `session-timeout` nor `timeoutSeconds` is specified, the `timeoutSeconds` default is used. Note that the `session-timeout` element in `web.xml` is specified in minutes, not seconds.
`enableCookies`	`true`	Uses cookies for session tracking if set to `true`.
`enableURLRewriting`	`true`	Enables URL rewriting. This provides session tracking via URL rewriting when the browser does not accept cookies. You must also use an `encodeURL` or `encodeRedirectURL` call in the servlet or JSP.

`ssl`

Defines SSL processing parameters.

Superelements

security (`sun-acc.xml`)

Subelements

none

Attributes

The following table describes attributes for the `SSL` element.

Table 142. `ssl attributes`

Attribute	Default	Description
`cert-nickname`	`s1as`	(optional) The nickname of the server certificate in the certificate database or the PKCS#11 token. In the certificate, the name format is `tokenname:nickname`. Including the `tokenname:` part of the name in this attribute is optional.
`ssl2-enabled`	`false`	(optional) Determines whether SSL2 is enabled.
`ssl2-ciphers`	none	(optional) A comma-separated list of the SSL2 ciphers to be used. Ciphers not explicitly listed will be disabled for the target, even if those ciphers are available in the particular cipher suite you are using. If this option is not used, all supported ciphers are assumed to be enabled. Allowed values are `rc4`, `rc4export`, `rc2`, `rc2export`, `idea`, `des`, `desede3`.
`ssl3-enabled`	`true`	(optional) Determines whether SSL3 is enabled.

Attribute	Default	Description
`ssl3-tls-ciphers`	none	(optional) A comma-separated list of the SSL3 and/or TLS ciphers to be used. Ciphers not explicitly listed will be disabled for the target, even if those ciphers are available in the particular cipher suite you are using. If this option is not used, all supported ciphers are assumed to be enabled. Allowed values are `SSL_RSA_WITH_RC4_128_MD5`, `SSL_RSA_WITH_3DES_EDE_CBC_SHA`, `SSL_RSA_WITH_DES_CBC_SHA`, `SSL_RSA_EXPORT_WITH_RC4_40_MD5`, `SSL_RSA_WITH_NULL_MD5`, `SSL_RSA_WITH_RC4_128_SHA`, `SSL_RSA_WITH_NULL_SHA`. Values available in previous releases are supported for backward compatibility.
`tls-enabled`	`true`	(optional) Determines whether TLS is enabled.
`tls-rollback-enabled`	`true`	(optional) Determines whether TLS rollback is enabled. Enable TLS rollback for Microsoft Internet Explorer 5.0 and 5.5.

`steady-pool-size`

Specifies the initial and minimum number of bean instances that are maintained in the pool. Default is 32. Applies to stateless session beans and message-driven beans.

Superelements

bean-pool (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`store-properties`

Specifies session persistence (storage) properties.

Superelements

session-manager (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `store-properties` element.

Table 143. `store-properties` Subelements

Element	Required	Description
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Properties

The following table describes properties for the `store-properties` element.

Table 144. `store-properties` Properties

Property	Default	Description
`directory`	`domain-dir/generated/jsp/app-name/module-name_war`	Specifies the absolute or relative pathname of the directory into which individual session files are written. A relative path is relative to the temporary work directory for this web module. Applicable only if the `persistence-type` attribute of the parent session-manager element is `file`.

Property	Default	Description
`persistenceScope`	`session`	Specifies how much of the session state is stored. Allowed values are as follows: • `session` - The entire session state is stored every time. This mode provides the best guarantee that your session data is correctly stored for any distributable web module. • `modified-session` - The entire session state is stored if it has been modified. A session is considered to have been modified if `HttpSession.setAttribute()` or `HttpSession.removeAttribute()` was called. You must guarantee that `setAttribute` is called every time an attribute is changed. This is not a Jakarta EE specification requirement, but it is required for this mode to work properly. • `modified-attribute` - Only modified session attributes are stored. For this mode to work properly, you must follow some guidelines, which are explained immediately following this table. Applicable only if the `persistence-type` attribute of the parent session-manager element is `hazelcast`.

If the `persistenceScope` store property is set to `modified-attribute`, a web module must follow these guidelines:

- Call `setAttribute` every time the session state is modified.
- Make sure there are no cross-references between attributes. The object graph under each distinct attribute key is serialized and stored separately. If there are any object cross references between the objects under each separate key, they are not serialized and deserialized correctly.
- Distribute the session state across multiple attributes, or at least between a read-only attribute and a modifiable attribute.

`stub-property`

Specifies JAX-RPC property values that are set on a `javax.xml.rpc.Stub` object before it is returned to the web service client. The property names can be any properties supported by the JAX-RPC `Stub` implementation.

Superelements

port-info (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml``)

Subelements

The following table describes subelements for the `stub-property` element.

Table 145. `stub-property` subelements

Element	Required	Description
name	only one	Specifies the name of the entity.
value	only one	Specifies the value of the entity.

Properties

The following table describes properties for the `stub-property` element.

Table 146. `stub-property` properties

Property	Default	Description
`jbi-enabled`	`true`	Determines whether the visibility of this endpoint as a Java Business Integration service is enabled or disabled.

Example

```
<service-ref>
<service-ref-name>service/FooProxy</service-ref-name>
  <port-info>
       <service-endpoint-interface>a.FooPort</service-endpoint-interface>
       <wsdl-port>
          <namespaceURI>urn:Foo</namespaceURI>
          <localpart>FooPort</localpart>
```

```
        </wsdl-port>
        <stub-property>
           <name>javax.xml.rpc.service.endpoint.address</name>
           <value>http://localhost:8080/a/Foo</value>
        </stub-property>
   </port-info>
</service-ref>
```

`target-server`

Specifies the IIOP listener for the target server. Also specifies IIOP endpoints used for load balancing. If the Payara Server instance on which the application client is deployed participates in a cluster, Payara Server finds all currently active IIOP endpoints in the cluster automatically. However, a client should have at least two endpoints specified for bootstrapping purposes, in case one of the endpoints has failed.

A listener or endpoint is in the form `host:port`, where the host is an IP address or host name, and the port specifies the port number.

Superelements

client-container (`sun-acc.xml`)

Subelements

The following table describes subelements for the `target-server` element.

Table 147. `target-server` subelements

Element	Required	Description
description	zero or one	Specifies the description of the target server.
security	zero or one	Specifies the security configuration for the IIOP/SSL communication with the target server.

Attributes

The following table describes attributes for the `target-server` element.

Table 148. `target-server` attributes

Attribute	Default	Description
`name`	none	Specifies the name of the server instance accessed by the client container.
`address`	none	Specifies the host name or IP address (resolvable by DNS) of the server to which this client attaches.
`port`	none	Specifies the naming service port number of the server to which this client attaches. For a new server instance, assign a port number other than 3700. You can change the port number in the Administration Console. Click the Help button in the Administration Console for more information.

`tie-class`

Specifies the automatically generated name of a tie implementation class for a port component.

Superelements

webservice-endpoint (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`timeout`

Specifies the cache-mapping specific maximum amount of time in seconds that an entry can remain in the cache after it is created or refreshed. If not specified, the default is the value of the `timeout` attribute of the cache element.

Superelements

cache-mapping (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none - contains data

Attributes

The following table describes attributes for the `timeout` element.

Table 149. `timeout` Attributes

Attribute	Default	Description
`name`	none	Specifies the timeout input parameter, whose value is interpreted in seconds. The field's type must be `java.lang.Long` or `java.lang.Integer`.
`scope`	`request.attribute`	(optional) Specifies the scope from which the input parameter is retrieved. Allowed values are `context.attribute`, `request.header`, `request.parameter`, `request.cookie`, `request.attribute`, and `session.attribute`.

`transport-config`

Specifies the security transport information.

Superelements

ior-security-config (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `transport-config` element.

Table 150. `transport-config` Subelements

Element	Required	Description
integrity	only one	Specifies if the target supports integrity-protected messages. The values are NONE, SUPPORTED, or REQUIRED.
confidentiality	only one	Specifies if the target supports privacy-protected messages. The values are NONE, SUPPORTED, or REQUIRED.

Element	Required	Description
establish-trust-in-target	only one	Specifies if the target is capable of authenticating to a client. The values are NONE, SUPPORTED, or REQUIRED.
establish-trust-in-client	only one	Specifies if the target is capable of authenticating a client. The values are NONE, SUPPORTED, or REQUIRED.

`transport-guarantee`

Specifies that the communication between client and server is `NONE`, `INTEGRAL`, or `CONFIDENTIAL`.

- `NONE` means the application does not require any transport guarantees.
- `INTEGRAL` means the application requires that the data sent between client and server be sent in such a way that it can't be changed in transit.
- `CONFIDENTIAL` means the application requires that the data be transmitted in a fashion that prevents other entities from observing the contents of the transmission.

In most cases, a value of `INTEGRAL` or `CONFIDENTIAL` indicates that the use of SSL is required.

Superelements

webservice-endpoint (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`unique-id`

Contains the unique ID for the application. This value is automatically updated each time the application is deployed or redeployed. Do not edit this value.

Superelements

payara-application (`glassfish-application.xml` (deprecated) and `payara-application.xml`), enterprise-beans (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`url-pattern`

Specifies a servlet URL pattern for which caching is enabled. See the Servlet 6.0 specification section 12.2 for applicable patterns.

Superelements

cache-mapping (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

none - contains data

`user-group`

Contains the group to which the principal belongs.

Superelements

security-map (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

none - contains data

`use-thread-pool-id`

Specifies the thread pool from which threads are selected for remote invocations of this bean.

Superelements

ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`value`

Specifies the value of the entity.

Superelements

call-property, stub-property (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml``); property-with-subelements (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`valve`

Specifies a custom valve for this web application. You can define a valve for all the web applications on a specific virtual server. For details, see create-virtual-server.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`)

Subelements

The following table describes subelements for the `valve` element.

Table 151. `valve` Subelements

Element	Required	Description
description	zero or one	Specifies a text description of this element.
property-with-attributes	zero or more	Specifies a property, which has a name and a value.

Attributes

The following table describes attributes for the `valve` element.

Table 152. `valve` Attributes

Attribute	Default	Description
`name`	none	Specifies a unique name for the valve.

Attribute	Default	Description
`class-name`	none	Specifies the fully qualified class name of the valve. The valve class must implement the `org.apache.catalina.Valve` interface from Tomcat or previous Payara Server releases, or the `org.glassfish.web.valve.GlassFishValve` interface from the current Payara Server release.

Example

```
<valve name="MyValve" classname="org.glassfish.extension.Valve">
   <property name="MyProperty1" value="MyValue1" />
   <property name="MyProperty2" value="MyValue2" />
</valve>
```

`vendor`

Specifies a vendor-specific icon, splash screen, text string, or a combination of these for Java Web Start download and launch screens. The complete format of this element's data is as follows:

```
<vendor>icon-image-URI::splash-screen-image-URI::vendor-text</vendor>
```

The following example vendor element contains an icon, a splash screen, and a text string:

```
<vendor>images/icon.jpg::otherDir/splash.jpg::MyCorp, Inc.</vendor>
```

The following example vendor element contains an icon and a text string:

```
<vendor>images/icon.jpg::MyCorp, Inc.</vendor>
```

The following example vendor element contains a splash screen and a text string; note the initial double colon:

```
<vendor>::otherDir/splash.jpg::MyCorp, Inc.</vendor>
```

The following example vendor element contains only a text string:

```
<vendor>MyCorp, Inc.</vendor>
```

The default value is the text string `Application Client`.

Superelements

java-web-start-access (`glassfish-application-client.xml` (deprecated) and `payara-application-client.xml`)

Subelements

none - contains data

`version-identifier`

Contains version information for an application or module. For more information about application versioning, see module-and-application-versions.

Superelements

payara-application (<code>glassfish-application.xml</code> (deprecated) and <code>payara-application.xml<code>), payara-web-app (<code>payara-web.xml</code>), payara-ejb-jar (<code>glassfish-ejb-jar.xml</code> (deprecated) and <code>payara-ejb-jar.xml</code>), payara-application-client (<code>glassfish-application-client.xml</code> (deprecated) and `payara-application-client.xml</code>)

Subelements

none - contains data

`victim-selection-policy`

Specifies how stateful session beans are selected for passivation. Possible values are First In, First Out (`FIFO`), Least Recently Used (`LRU`), Not Recently Used (`NRU`). The default value is `NRU`, which is actually pseudo-LRU.

NOTE	You cannot plug in your own victim selection algorithm.

The victims are generally passivated into a backup store (typically a file system or database). This store is cleaned during startup, and also by a periodic background process that removes idle entries as specified by `removal-timeout-in-seconds`.

The backup store is monitored by a background thread (or sweeper thread) to remove unwanted entries.

Applies to stateful session beans.

Superelements

bean-cache (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

Example

```
<victim-selection-policy>LRU</victim-selection-policy>
```

If both SSL2 and SSL3 are enabled, the server tries SSL3 encryption first. If that fails, the server tries SSL2 encryption. If both SSL2 and SSL3 are enabled for a virtual server, the server tries SSL3 encryption first. If that fails, the server tries SSL2 encryption.

`web`

Specifies the application's web tier configuration.

Superelements

payara-application (`payara-application.xml`)

Subelements

The following table describes subelements for the `web` element.

Table 153. `web` Subelements

Element	Required	Description
web-uri	only one	Contains the web URI for the application.
context-root	only one	Contains the web context root for the web module.

`web-uri`

Contains the web URI for the application. Must match the corresponding element in the `application.xml` file.

Superelements

web (`glassfish-application.xml` (deprecated) and `payara-application.xml`)

Subelements

none - contains data

`webservice-default-login-config`

When declaring a secured Web Service based on an EJB using the `payara-ejb-jar.xml` deployment descriptor, it's necessary to define the login configuration (authentication method, security realm name, etc.) for each EJB Web Service that is secured inside the assembly.

For example, if an application contains 2 EJB web services called **EJBWS1** and **EJBWS2**, and they need to be secured using `BASIC` authentication against the *file* security realm, the following configuration would be needed:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-ejb-jar PUBLIC "-//Payara.fish//DTD Payara Application Server 7
EJB 4.0 Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-ejb-jar_4_0-1.dtd">
<payara-ejb-jar>
    <enterprise-beans>
        <ejb>
            <ejb-name>EJBWS1</ejb-name>
            <webservice-endpoint>
              <port-component-name>EJBWS1Port</port-component-name>
              <endpoint-address-uri>EJBWS1/EJBWebService</endpoint-address-uri>
              <login-config>
                <auth-method>BASIC</auth-method>
                <realm>file</realm>
              </login-config>
            </webservice-endpoint>
          </ejb>
        <ejb>
            <ejb-name>EJBWS2</ejb-name>
            <webservice-endpoint>
              <port-component-name>EJBWS2Port</port-component-name>
              <endpoint-address-uri>EJBWS2/EJBWebService</endpoint-address-uri>
              <login-config>
                <auth-method>BASIC</auth-method>
                <realm>file</realm>
              </login-config>
            </webservice-endpoint>
          </ejb>
    </enterprise-beans>
```

```
</payara-ejb-jar>
```

Notice that the `login-config` element is repeated exactly like it is in the 2 EJB definitions. Not only that, but if these Web services are defined using annotations for each EJB component, then the **JAX-WS** information (Port Component Name, Endpoint Address, etc.) would be duplicated too, which is too cumbersome for cases when there are lots of EJB Web service definitions.

For this scenario, the `webservice-default-login-config` has been introduced to simplify this configuration. When this element is declared, the login configuration inside it will apply to all the EJB defined Web Services by default.

The previous example can be simplified like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-ejb-jar PUBLIC "-//Payara.fish//DTD Payara Application Server 7
EJB 4.0 Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-ejb-jar_4_0-1.dtd">
<payara-ejb-jar>
  <webservice-default-login-config>
      <auth-method>BASIC</auth-method>
      <realm>file</realm>
  </webservice-default-login-config>
</payara-ejb-jar>
```

NOTE | All sub-elements tags of the `login-config` can be used inside this element.

IMPORTANT | If an EJB Web service definition needs a different login configuration from the default, just redefine it as shown in the example, and it will override the default configuration.

Superelements

payara-ejb-jar (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `webservice-default-login-`

`config` element.

Table 154. `webservice-default-login-config` *subelements*

Element	Required	Description
auth-method	only one	Specifies the authentication method.
realm	zero or one	Specifies the name of the realm used to process all authentication requests.

`webservice-description`

Specifies a name and optional publish location for a web service.

Superelements

payara-web-app (`glassfish-web.xml` (deprecated) and `payara-web.xml`), enterprise-beans (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `webservice-description` element.

Table 155. `webservice-description` *subelements*

Element	Required	Description
webservice-description-name	only one	Specifies a unique name for the web service within a web or EJB module.
wsdl-publish-location	zero or one	Specifies the URL of a directory to which a web service's WSDL is published during deployment.

`webservice-description-name`

Specifies a unique name for the web service within a web or EJB module.

Superelements

webservice-description (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

`webservice-endpoint`

Specifies information about a web service endpoint.

Superelements

servlet (`glassfish-web.xml` (deprecated) and `payara-web.xml`), ejb (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

The following table describes subelements for the `webservice-endpoint` element.

Table 156. `webservice-endpoint` subelements

Element	Required	Description
port-component-name	only one	Specifies a unique name for a port component within a web or EJB module.
endpoint-address-uri	zero or one	Specifies the automatically generated endpoint address.
login-config	zero or one	Specifies the authentication configuration for an EJB web service endpoint.
message-security-binding	zero or one	Specifies a custom authentication provider binding.
transport-guarantee	zero or one	Specifies that the communication between client and server is `NONE`, `INTEGRAL`, or `CONFIDENTIAL`.
service-qname	zero or one	Specifies the WSDL service element that is being referenced.
tie-class	zero or one	Specifies the automatically generated name of a tie implementation class for a port component.
servlet-impl-class	zero or one	Specifies the automatically generated name of the generated servlet implementation class.

Element	Required	Description
debugging-enabled	zero or one	Specifies whether the debugging servlet is enabled for this web service endpoint. Allowed values are `true` and `false` (the default).
property-with-attributes (`glassfish-web.xml` (deprecated) and `payara-web.xml`) property-with-subelements (`glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)	zero or more	Specifies a property, which has a name and a value.

`whitelist-package`

This element can be used to whitelist packages on extreme-classloading-isolation.

Whitelisted packages are taken into account by the server when scanning libraries.

For more information about how extreme class loading isolation works on Payara Server, see the enhanced-classloading section.

Superelements

payara-application (`glassfish-application.xml` (deprecated) and `payara-application.xml`), payara-web-app (`payara-web.xml`)

Subelements

none

`work-security-map`

Defines a work security map, which maps a principal associated with an incoming work instance to a principal in the Payara Server's security domain. It is possible to map multiple EIS group or user principals to the same Payara Server principal.

This is different from a security-map, which maps the principal received during servlet or EJB authentication to the credentials accepted by the EIS.

Superelements

resources (`glassfish-resources.xml` (deprecated) and `payara-resources.xml`)

Subelements

The following table describes subelements for the `work-security-map` element.

Table 157. `work-security-map` Subelements

Element	Required	Description
description	zero or one	Contains a text description of this element.
principal-map	zero or more	Maps an EIS principal to a principal defined in the Payara Server domain.
group-map	zero or more	Maps an EIS group to a group defined in the Payara Server domain.

Attributes

The following table describes attributes for the `work-security-map` element.

Table 158. `work-security-map` Attributes

Attribute	Default	Description
`name`	none	Specifies a unique name for the work security map.
`description`	none	Specifies a text description for this element.

`wsdl-override`

Specifies a valid URL pointing to a final WSDL document. If not specified, the WSDL document associated with the `service-ref` in the standard Jakarta EE deployment descriptor is used.

Superelements

service-ref (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml``)

Subelements

none - contains data

Example

```
<!-- available via HTTP -->
<wsdl-override>http://localhost:8000/myservice/myport?WSDL</wsdl-override>
```

```
<!-- in a file -->
<wsdl-override>file:/home/user1/myfinalwsdl.wsdl</wsdl-override>
```

`wsdl-port`

Specifies the WSDL port.

Superelements

port-info (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`, `glassfish-application-client.xml` (deprecated) and `payara-application-client.xml``)

Subelements

The following table describes subelements for the `wsdl-port` element.

Table 159. `wsdl-port` subelements

Element	Required	Description
namespaceuri	only one	Specifies the namespace URI.
localpart	only one	Specifies the local part of a QNAME.

`wsdl-publish-location`

Specifies the URL of a directory to which a web service's WSDL is published during deployment. Any required files are published to this directory, preserving their location relative to the module-specific WSDL directory (`META-INF/wsdl` or `WEB-INF/wsdl`).

Superelements

webservice-description (`glassfish-web.xml` (deprecated) and `payara-web.xml`, `glassfish-ejb-jar.xml` (deprecated) and `payara-ejb-jar.xml`)

Subelements

none - contains data

Example

Suppose you have an `ejb.jar` file whose `webservices.xml` file's `wsdl-file` element contains the following reference:

```
META-INF/wsdl/a/Foo.wsdl
```

Suppose your `glassfish-ejb.jar` file contains the following element:

```
<wsdl-publish-location>file:/home/user1/publish</wsdl-publish-location>
```

The final WSDL is stored in `/home/user1/publish/a/Foo.wsdl`.

High Availability

High Availability in Payara Server

This chapter describes the high availability features in Payara Server.

The following topics are addressed here:

- "Overview of High Availability"
- "How Payara Server Provides High Availability"
- "Recovering from Failures"

Overview of High Availability

High availability applications and services provide their functionality continuously, regardless of hardware and software failures. To make such reliability possible, Payara Server provides mechanisms for maintaining application state data between clustered Payara Server instances. Application state data, such as HTTP session data, stateful EJB sessions, and dynamic cache information, is replicated in real time across server instances. If any one server instance goes down, the session state is available to the next failover server, resulting in minimum application downtime and enhanced transactional security.

Payara Server provides the following high availability features:

- "Load Balancing With the Apache mod_jk or mod_proxy_ajp Module"
- "High Availability Session Persistence"
- "High Availability Java Message Service"
- "RMI-IIOP Load Balancing and Failover"

Load Balancing with the Apache `mod_jk` or `mod_proxy_ajp` Module

A common load balancing configuration for Payara Server is to use the Apache HTTP Server as the web server front-end, and the Apache `mod_jk` or `mod_proxy_ajp` module as the connector between the web server and Payara Server. See configuring-payara-server-with-apache-http-server-and-mod_jk and configuring-payara-server-with-apache-http-server-and-mod_proxy_ajp for more information.

High Availability Session Persistence

Payara Server provides high availability of HTTP requests and session data (both HTTP session data and stateful session bean data).

Jakarta EE applications typically have significant amounts of session state data. A web shopping cart is the classic example of a session state. Also, an application can cache frequently-needed data in the session object. In fact, almost all applications with significant user interactions need to maintain session state. Both HTTP sessions and stateful session beans (SFSBs) have session state data.

Preserving session state across server failures can be important to end users. If the Payara Server instance hosting the user session experiences a failure, the session state can be recovered, and the session can continue without loss of information. High availability is implemented in Payara Server by means of in-memory session replication on Payara Server instances running in a cluster.

For more information about in-memory session replication in Payara Server, see how-payara-server-provides-high-availability. For detailed instructions on configuring high availability session persistence, see configuring-high-availability-session-persistence-

and-failover.

High Availability Java Message Service

Payara Server supports the Java Message Service (JMS) API and JMS messaging through its built-in `jmsra` resource adapter communicating with Open Message Queue as the JMS provider. This combination is often called the **JMS Service**.

The JMS service makes JMS messaging highly available as follows:

Message Queue Broker Clusters: By default, when a Payara Server cluster is created, the JMS service automatically configures a Message Queue broker cluster to provide JMS messaging services, with one clustered broker assigned to each cluster instance. This automatically created broker cluster is configurable to take advantage of the two types of broker clusters, conventional and enhanced, supported by Message Queue.

+ Additionally, Message Queue broker clusters created and managed using Message Queue itself can be used as external, or remote, JMS hosts. Using external broker clusters provides additional deployment options, such as deploying Message Queue brokers on different hosts from the Payara instances they service, or deploying different numbers of Message Queue brokers and Payara instances.

+ For more information about Message Queue clustering, see using-message-queue-broker-clusters-with-payara-server.

Connection Failover: The use of Message Queue broker clusters allows connection failover in the event of a broker failure. If the primary JMS host (Message Queue broker) in use by a Payara Server instance fails, connections to the failed JMS host will automatically fail over to another host in the JMS host list, allowing messaging operations to continue and maintaining JMS messaging semantics.

+ For more information about JMS connection failover, see connection-failover.

RMI-IIOP Load Balancing and Failover

With RMI-IIOP load balancing, IIOP client requests are distributed to different server

instances or name servers, which spreads the load evenly across the cluster, providing scalability. IIOP load balancing combined with EJB clustering and availability also provides EJB failover.

When a client performs a JNDI lookup for an object, the Naming Service essentially binds the request to a particular server instance. From then on, all lookup requests made from that client are sent to the same server instance, and thus all `EJBHome` objects will be hosted on the same target server. Any bean references obtained henceforth are also created on the same target host. This effectively provides load balancing, since all clients randomize the list of target servers when performing JNDI lookups. If the target server instance goes down, the lookup or EJB method invocation will fail over to another server instance.

IIOP Load balancing and failover happens transparently. No special steps are needed during application deployment. If the Payara Server instance on which the application client is deployed participates in a cluster, the Payara Server finds all currently active IIOP endpoints in the cluster automatically. However, a client should have at least two endpoints specified for bootstrapping purposes, in case one of the endpoints has failed.

For more information on RMI-IIOP load balancing and failover, see rmi-iiop-load-balancing-and-failover.

How Payara Server Provides High Availability

Payara Server provides high availability through the following subcomponents and features:

- storage-for-session-state-data
- highly-available-clusters

Storage for Session State Data

Storing session state data enables the session state to be recovered after the failover of a server instance in a cluster. Recovering the session state enables the session to continue without loss of information. Payara Server supports in-memory session

replication on other servers in the cluster for maintaining HTTP session and stateful session bean data.

In-memory session replication is implemented in Payara Server via the use of the Data Grid (implemented using Hazelcast Community Edition). As such, to use this feature the Data Grid has to be enabled in the domain first.

Highly Available Clusters

A highly available cluster integrates a state replication service with clusters and load balancer.

IMPORTANT	When implementing a highly available cluster solution, use a load balancer that includes session-based stickiness as part of its load-balancing algorithm. Otherwise, session data can be misdirected or lost.

Clusters, Instances, Sessions, and Load Balancing

Clusters, server instances, load balancers, and sessions are related as follows:

- A server instance is not required to be part of a cluster. However, an instance that is not part of a cluster cannot take advantage of high availability through transfer of session state from one instance to other instances.
- The server instances within a cluster can be hosted on one or multiple hosts. You can group server instances across different hosts into a cluster.
- A particular load balancer can forward requests to server instances on multiple clusters. You can use this ability of the load balancer to perform an online upgrade without loss of service.
- A single cluster can receive requests from multiple load balancers. If a cluster is served by more than one load balancer, you must configure the cluster in exactly the same way on each load balancer.
- Each session is tied to a particular cluster. Therefore, although you can deploy an application on multiple clusters, session failover will occur only within a single

cluster.

The cluster thus acts as a safe boundary for session failover for the server instances within the cluster. You can use the load balancer and upgrade components within the Payara Server without loss of service.

Protocols for Centralized Cluster Administration

Payara Server uses the secure shell (SSH) to ensure that clusters that span multiple hosts can be administered centrally. To perform administrative operations on Payara Server instances that are remote from the domain administration server (DAS), the DAS must be able to communicate with those instances. If an instance is running, the DAS connects to the running instance directly. For example, when you deploy an application to an instance, the DAS connects to the instance and deploys the application to the instance.

However, the DAS cannot connect to an instance to perform operations on an instance that is not running, such as creating or starting the instance. For these operations, the DAS uses SSH to contact a remote host and administer instances there. SSH provides confidentiality and security for data that is exchanged between the DAS and remote hosts.

TIP	The use of SSH to enable centralized administration of remote instances is optional. If the use of SSH is not feasible in your environment, you can administer remote instances locally.

For more information, see enabling-centralized-administration-of-payara-server-instances.

Recovering from Failures

You can use various techniques to manually recover individual subcomponents after hardware failures such as disk crashes.

The following topics are addressed here:

- "Recovering the Domain Administration Server"
- "Recovering Payara Server Instances"
- "Recovering Message Queue"

Recovering the Domain Administration Server

Loss of the Domain Administration Server (DAS) affects only administration. Payara Server clusters and standalone instances, and the applications deployed to them, continue to run as before, even if the DAS is not reachable

Use any of the following methods to recover the DAS:

- Back up the domain periodically, so you have periodic snapshots. After a hardware failure, re-create the DAS on a new host, as described in "re-creating-the-domain-administration-server-das" in Payara Server General Administration section.
- Put the domain installation and configuration on a shared and robust file system (NFS for example). If the primary DAS host fails, a second host is brought up with the same IP address and will take over with manual intervention or user supplied automation.
- Zip the Payara Server installation and domain root directory. Restore it on the new host, assigning it the same network identity.

Recovering Payara Server Instances

Payara Server provide tools for backing up and restoring Payara Server instances. For more information, see to-resynchronize-an-instance-and-the-das-offline.

Recovering Message Queue

When a Message Queue broker becomes unavailable, the method you use to restore the broker to operation depends on the nature of the failure that caused the broker to become unavailable:

- Power failure or failure other than disk storage
- Failure of disk storage

Additionally, the urgency of restoring an unavailable broker to operation depends on the type of the broker:

Standalone Broker: When a standalone broker becomes unavailable, both service availability and data availability are interrupted. Restore the broker to operation as soon as possible to restore availability. **Broker in a Conventional Cluster**: When a broker in a conventional cluster becomes unavailable, service availability continues to be provided by the other brokers in the cluster. However, data availability of the persistent data stored by the unavailable broker is interrupted. Restore the broker to operation to restore availability of its persistent data. **Broker in an Enhanced Cluster**: When a broker in an enhanced cluster becomes unavailable, service availability and data availability continue to be provided by the other brokers in the cluster. Restore the broker to operation to return the cluster to its previous capacity.

Recovering from Power Failure and Failures Other than Disk Storage

When a host is affected by a power failure or failure of a non-disk component such as memory, processor or network card, restore Message Queue brokers on the affected host by starting the brokers after the failure has been remedied.

To start brokers serving as `EMBEDDED` or `LOCAL` JMS hosts, start the Payara instances the brokers are servicing.

To start brokers serving as `REMOTE` JMS hosts, use the `imqbrokerd` Message Queue utility.

Recovering from Failure of Disk Storage

Message Queue uses disk storage for software, configuration files and persistent data stores. In a default Payara Server installation, all three of these are generally stored on the same disk: the Message Queue software in `as-install-parent/mq`, and broker configuration files and persistent data stores (except for the persistent data stores of enhanced clusters, which are housed in highly available databases) in `domain-dir/imq`. If this disk fails, restoring brokers to operation is impossible unless you have previously created a backup of these items.

To create such a backup, use a utility such as `zip`, `gzip` or `tar` to create archives of these directories and all their content. When creating the backup, you should first quiesce all brokers and physical destinations, as described in the Open Message Queue Administration Guide, respectively. Then, after the failed disk is replaced and put into service, expand the backup archive into the same location.

Restoring the Persistent Data Store from Backup.

For many messaging applications, restoring a persistent data store from backup does not produce the desired results because the backed up store does not represent the content of the store when the disk failure occurred. In some applications, the persistent data changes rapidly enough to make backups obsolete as soon as they are created.

To avoid issues in restoring a persistent data store, consider using a RAID or SAN data storage solution that is fault-tolerant, especially for data stores in production environments.

See Also

For information about planning a high-availability deployment, including assessing hardware requirements, planning network configuration, and selecting a topology, see the "Payara Server Deployment Planning section". This manual also provides a high-level introduction to concepts such as:

- Payara Server components such as node agents, domains, clusters and deployment groups
- IIOP load balancing in a cluster or deployment group
- Message queue fail-over

For more information about developing applications that take advantage of high availability features, see the Payara Server Application Development section.

Enabling Centralized Administration of Payara Server Instances

IMPORTANT	At the moment of the publication of this documentation, using the natively provided SSH utilities by Windows OS environments

> doesn't work. The main reason for this is significant issues with file transfers not working correctly across nodes.
>
> We have discovered that when SSH is provided using a Cygwin-based OpenSSH and Bash shell from Cygwin is used to manage Payara Server, remote nodes can be created and operated without issues.
>
> Until the problems with natively provided OpenSSH implementation are addressed, we recommend you configure SSH nodes using the Cygwin-based OpenSSH toolset in production environments.

Payara Server uses the secure shell (SSH) to ensure that instances that span multiple hosts can be administered centrally.

To perform administrative operations on Payara Server instances that are remote from the domain administration server (DAS), the DAS must be able to communicate with those instances.

If an instance is running, the DAS connects to the running instance directly. For example, when you deploy an application to an instance, the DAS connects to the instance and deploys the application to the instance.

However, the DAS cannot connect to an instance to perform operations on an instance that is not running, such as creating or starting the instance.

For these operations, the DAS uses SSH to contact a remote host and administer instances there. SSH provides confidentiality and security for data that is exchanged between the DAS and remote hosts.

About Centralized Administration of Payara Server Instances

The use of SSH to enable centralized administration of remote instances is optional and is required only for specific operations. Instances local to the DAS can be administered

without SSH. If SSH is not a possibility in your environment, you can administer remote instances locally instead.

Determining Whether to Enable Centralized Administration

Before setting up a Payara Server cluster or deployment group, use the following considerations to determine whether to enable centralized administration of remote instances:

- If you are planning to set up a large cluster or deployment group of many instances, consider enabling centralized administration of their corresponding instances.

  Centralized administration of instances simplifies the administration of the Deployment Groups by enabling you to perform all administrative operations on the Deployment Groups and its instances from the DAS.

- If you are planning a small Deployment Groups of few instances, consider whether enabling centralized administration requires more effort than logging in to individual hosts to administer remote instances locally.

How you administer instances and the nodes on which they reside varies depending on whether and how centralized administration is enabled. The following table provides cross-references to instructions for administering nodes and instances depending on the protocol that is used for enabling centralized administration, if any.

Protocol	Node Administration Instructions	Instance Administration Instructions
administering-payara-server-instances-centrally	SSH	creating-listing-testing-and-deleting-ssh-nodes
administering-payara-server-instances-centrally	None	creating-listing-and-deleting-config-nodes

Protocol	Node Administration Instructions	Instance Administration Instructions
administering -payara-server -instances -locally		

Considerations for Using SSH for Centralized Administration

In a typical Payara Server deployment, the DAS acts as the SSH client, and hosts where instances reside act as SSH servers. The SSH Server Daemon `sshd` must be running on hosts where instances reside, but is not required to be running on the DAS host.

The DAS uses its own SSH client for communicating with hosts where instances reside. However, to generate keys and test SSH setup, a native SSH client must be installed on the DAS host.

The requirements for SSH configuration and user management are different for each operating system on which Payara Server is supported. Therefore, the use of SSH for centralized administration involves using SSH tools to configure SSH on the operating system that you are using.

On UNIX and Linux systems, SSH is typically installed and preconfigured, and requires minimal additional setup. On Windows systems, additional setup is required to install and configure an SSH provider.

Obtaining SSH Software

On UNIX and Linux systems, SSH software is typically installed as part of the base operating system.

However, on Windows systems, you must install one of the following SSH providers:

- Cygwin
- MKS Toolkit for Developers

Determining the SSH User

Before setting up SSH, decide which SSH user Payara Server will use when connecting to remote hosts. For the following reasons, administration is simplest if the SSH user is the user that starts the DAS:

- For public key authentication, the user that starts the DAS must be able to read the SSH user's private key file.
- Remote instances are started as the SSH user.
- By default, the DAS assumes that the SSH user is the user that is running the DAS.

Requirements for the SSH User's Environment

The SSH user's environment on a host is set by the environment set-up files that are run when the user uses SSH to run a command on the host. You must ensure that these files set up the SSH user's environment correctly.

The files that are run when the user uses SSH to run a command are different from the files that are run when the user logs in to a host. For example, in the bash shell, `.profile` and `.bashrc` are run when the user logs in, but only `.bashrc` is run when the user runs a command.

Therefore, in the bash shell, you must ensure that `.bashrc` contains the required environment settings for the SSH user.

File Access Permissions on UAC-Enabled Windows Systems

WARNING

The User Account Control (UAC) (`http://technet.microsoft.com/en-us/library/cc709691%28WS.10%29.aspx`) feature is available only on older versions of the Windows operating system(like Windows 7, Windows Vista, and Windows Server 2008).

You might be using a UAC-enabled Windows system and choose to store files for Payara Server instances in a directory other than the SSH user's home directory. In this situation, the SSH user must have native (that is, non-virtual) read and write access to

the file system where the instances are to be stored. The OS-level administrator has such access by default.

You can also configure the system to grant such access to other users. For more information, see the documentation for the Windows operating system.

Setting up Cygwin SSH on Windows

Set up Cygwin SSH on the DAS host and on all hosts where instances in your Deployment Groups or clusters will reside.

Download and Install Cygwin

For centralized Payara Server administration, a basic Cygwin installation that includes the SSH client and the SSH server daemon `sshd` is sufficient. The default installation options are sufficient to create such a basic installation.

1. Log in as a user with Administrator privileges.
2. Create the `C:\cygwin` directory.
3. From the Cygwin home page (`http://www.cygwin.com/`), download and save the `setup.exe` file to your desktop.
4. Run the `setup.exe` file.
5. Select the default for the following options:
    - Install from Internet
    - Install Root Directory: `C:\cygwin`
    - Install for All Users
6. Specify a folder for the local package directory that is not the Cygwin root folder, for example, `C:\cygwin\packages`.
7. Specify the connection method.

   For example, if the host is connected to the Internet through a proxy server, specify the proxy server.

8. Select the mirror site from which to download the software.
9. Select the `openssh` package for installation.
10. Under the *Net* category, search for `openssh`.
11. Locate the `openssh` package and click *Skip*.

    The package is selected.

12. Click Next.

    Any unsatisfied dependencies are listed.

13. Leave the Select Required Packages option selected and click Next

    The packages are installed.

14. Click Finish.

See Also

For detailed information about installing Cygwin, see "Internet Setup" in Cygwin User's Guide (`http://cygwin.com/cygwin-ug-net/setup-net.html#internet-setup`).

To Set the Path for Windows and for the Cygwin Shell

To enable Payara Server tools to find commands for SSH, each user's path for Windows and for the Cygwin shell must contain the following directories:

- The Cygwin `bin` directory, for example `C:\cygwin\bin`
- The `bin` directory of the JDK software

1. Log in as a user with Administrator privileges.

   Logging in as a user with Administrator privileges ensures that the change applies to all users.

2. In the *System Information* control panel, click *Advanced* → *Environment Variables*.

3. Add the following directories to the `PATH` environment variable:

- The Cygwin `bin` directory, for example `C:\cygwin\bin`
- The `bin` directory of the JDK software installation

To Set the Home Directory for the Cygwin SSH User

The SSH Server Daemon `sshd` locates a user's home directory from the configuration in the user database, not from environment variables such as `HOME`. To ensure that all Payara Server commands can run without errors, each SSH user must be configured to have a home directory.

Each user on a Windows host where SSH is set up potentially has two home directories:

Windows home directory: Payara Server commands, which are run in a Windows command window, use the Windows home directory. **SSH home directory**: SSH commands, which are run in a shell such as `bash` or `ksh`, use the SSH home directory.

If these home directories are different, Payara Server and SSH each locate a user's `.ssh` directory in different directories. To simplify the setup of SSH, configure each user's home directory for SSH and Windows to be the same directory. A disadvantage of this approach is that the SSH home directory has spaces in its path name. Spaces in path names are cumbersome in the UNIX environment.

1. Log in as a user with Administrator privileges.
2. In the `c:\cygwin\etc\passwd` file, edit the home directory setting for the SSH user to specify the user's home directory for Windows.

To Configure and Start the Cygwin SSH Server Daemon `sshd`

Before You Begin

Ensure that the following prerequisites are met:

- A user account is created for each user that will log in to the host through SSH.
- A password is set for each user account.

The SSH server daemon `sshd` disallows authentication of any user for whose account a password is not set. . Double-click the Cygwin icon.

+ A Cygwin terminal is started.

1. If necessary, set the password for your user account.
   a. Run the `passwd` command as follows:

   ```
   $ passwd user-name
   ```

   user-name: The username for your account.

   b. Type a password.

   The password for your Windows account is also set.

2. Configure SSH on the host.
   a. Run the `ssh-host-config` command.

   ```
   $ ssh-host-config
   ```

   > **TIP** If you are using Windows XP, specify the `-y` option of `ssh-host-config` to answer `yes` to all prompts. If you run `ssh-host-config` with the `-y` option, omit Step **b**.

   b. Ensure that the `StrictModes` and `PubkeyAuthentication` options are set to `yes` in the file `/etc/ssh_config`.

   The file `/etc/ssh_config` can also be accessed as `/cygdrive/c/cygwin/etc/sshd_config`.

3. Start the SSH server daemon `sshd`.

   ```
   $ net start sshd
   ```

4. Confirm that the SSH server daemon `sshd` is running.

```
$ cygrunsrv --query sshd
 Service             : sshd
 Display name        : CYGWIN sshd
 Current State       : Running
 Controls Accepted   : Stop
 Command             : /usr/sbin/sshd -D
```

Next Steps

After you have completed the setup of SSH on a host, test the setup on the host as explained in testing-the-ssh-setup-on-a-host.

Setting up the MKS Toolkit on Windows

Set up the MKS Toolkit on the DAS host and on all hosts where instances in your Deployment Groups will reside.

To Install the MKS Toolkit

For centralized Payara Server administration, the default installation of the MKS Toolkit is sufficient.

Follow the instructions in the MKS Toolkit product documentation to install OpenSSH from the MKS Toolkit with default installation options.

See Also

For detailed information about installing MKS Toolkit, see "Installing MKS Toolkit" in the MKS Toolkit v9.4 Release Notes (`http://www.mkssoftware.com/docs/rn/relnotes_tk94.asp#install`).

To Set the Path for Windows and for the MKS Toolkit Shell

To enable Payara Server tools to find commands for SSH, each user's path for Windows and for the MKS Toolkit shell must contain the following directories:

- The MKS Toolkit `bin` directory, for example `C:\Program Files\MKS Toolkit\mksnt`

- The `bin` directory of the JDK software

The MKS Toolkit installer automatically adds the MKS Toolkit `bin` directory to the path. However, you must add the `bin` directory of the JDK software to the path yourself.

1. Log in as a user with Administrator privileges.

   Logging in as a user with Administrator privileges ensures that the change applies to all users.

2. In the System Information control panel, click Advanced>Environment Variables.
3. Add the `bin` directory of the JDK software to the Path environment variable.

To Set the Home Directory for the MKS Toolkit SSH User

The SSH Server Daemon `sshd` locates a user's home directory from the configuration in the user database, not from environment variables such as `HOME`. To ensure that all Payara Server commands can run without errors, each SSH user must be configured to have a home directory.

Each user on a Windows host where SSH is set up potentially has two home directories:

- Windows home directory. Payara Server commands, which are run in a Windows command window, use the Windows home directory.
- SSH home directory. SSH commands, which are run in a shell such as `bash` or `ksh`, use the SSH home directory.

If these home directories are different, Payara Server and SSH each locate a user's `.ssh` directory in different directories. To simplify the setup of SSH, configure each user's home directory for SSH and Windows to be the same directory.

A disadvantage of this approach is that the SSH home directory has spaces in its path name. Spaces in path names are cumbersome in a UNIX environment.

1. Compare the pairs of settings for Windows and the MKS Toolkit that are listed in the following table.

Windows Environment Variable	MKS Toolkit Field
`HOMEPATH`	Home Directory
`HOMEDRIVE`	Home Directory Drive

a. In a Windows command window, determine the values of the `HOMEPATH` and `HOMEDRIVE` environment variables.

b. In an MKS Toolkit shell, determine the current settings of the Home Directory and Home Directory Drive fields for the user.

```
$ userinfo user-name
```

user-name: The username for the user whose home directory you are setting, for example `Administrator`.

2. If the settings do not match, update setting of each MKS Toolkit field to match its corresponding Windows environment variable.

   If the settings match, no further action is required.

   To update the settings, run the following command in an MKS Toolkit shell:

```
$ userinfo -u -fHomeDirDrive:"drive" -fHomeDir:"path" user-name
```

   drive: The drive identifier of the disk drive on which the user's Windows home directory resides, for example, `C:`. **path**: The path to the user's Windows home directory, for example, `\Documents and Settings\Administrator`. **user-name**: The username for the user whose home directory you are setting, for example `Administrator`.

   NOTE | Do not set the `HOME` environment variable explicitly. If the Home Directory and Home Directory Drive are set correctly, the `HOME` environment variable specifies the correct path by default.

3. In an MKS Toolkit shell, confirm that the settings were updated.

```
$ userinfo user-name
```

user-name: The username for the user whose home directory you are setting, for example `Administrator`.

4. Log out of the host and log in to the host again.
5. Confirm that the home directories are the same as explained in Step **1**.

The following example sets the home directory for the MKS Toolkit user `Administrator` to `C:\Documents and Settings\Administrator`.

```
$ userinfo -u -fHomeDirDrive:"C:" -fHomeDir:"\Documents and Settings\Administrator"
Administrator
```

To Configure and Start the MKS Toolkit SSH Server Daemon `sshd`

NOTE Do not set the command shell to `cmd.exe`. The use of SSH for centralized Payara Server administration requires a shell in the style of a UNIX shell.

1. From the Programs menu, choose *MKS Toolkit* → *Configuration* → *Configuration Information*.
2. Enable password authentication and strict modes.
   a. Click the *Secure Shell Service* tab.
   b. Select the *Password Authentication* option.
   c. Click *Advanced* settings.
   d. Click the *Login* tab.
   e. Deselect the *Strict Modes* option.
3. If you are using SSH key-file authentication, enable `MKSAUTH` password authentication.

a. Click the Authentication tab.

b. Under *Enable/Disable Password* using `MKSAUTH`, type the user's password and click the Enable.

4. Start the SSH server daemon `sshd`.

   a. Confirm that the SSH server daemon `sshd` is running.

```
$ service query MKSSecureSH

Name:           MKS Secure Shell Service
Service Type:   WIN32_OWN_PROCESS
Current State:  RUNNING
Controls Accepted:      ACCEPT_STOP
Check Point:    0
Wait Hint:      0
Start Type:     AUTO_START
Error Control:  IGNORE
Path:           "C:\Program Files\MKS Toolkit\bin\secshd.exe"
Dependency:     NuTCRACKERService
Dependency:     tcpip
Service Start Name:     LocalSystem
```

Next Steps

After you have completed the setup of SSH on a host, test the setup on the host as explained in testing-the-ssh-setup-on-a-host.

Setting up SSH on UNIX and Linux Systems

Setting up SSH on UNIX and Linux systems involves verifying that the SSH server daemon `sshd` is running and, if necessary, starting this daemon. Set up SSH on the DAS host and on all hosts where instances in your Deployment Groups will reside.

On UNIX and Linux systems, SSH software is typically installed as part of the base operating system. If SSH is not installed, download and install the appropriate OpenSSH (`http://www.openssh.com/`) SSH package for your operating system.

To Set up SSH on Oracle Solaris Systems

1. Ensure that the following options in the configuration file `/etc/ssh/sshd_config` are set to `yes`:

- `StrictModes`
- `PubkeyAuthentication`

2. Determine if the SSH server daemon `sshd` is running.

```
$ /usr/bin/svcs ssh
```

3. If the SSH server daemon `sshd` is not running, start this daemon.

   If the daemon is running, no further action is required.

```
$ /usr/sbin/svcadm enable ssh
```

The following example confirms that the SSH server daemon `sshd` is running on an Oracle Solaris system.

```
$ /usr/bin/svcs ssh
STATE          STIME    FMRI
online         Jul_06   svc:/network/ssh:default
```

See Also

`svcs`(1)

Next Steps

After you have completed the setup of SSH on a host, test the setup on the host as explained in testing-the-ssh-setup-on-a-host.

To Set up SSH on Macos Systems

1. Open *System Preferences* and click *Sharing*.

   The Sharing window opens.

2. Ensure that *Remote Login* is selected in the Service list.
3. Ensure that either of the following is allowed access:

- All Users
- The user that running the DAS or instance

4. (macOS 10.6 systems only) Ensure that the SSH server daemon `sshd` allows password authentication.

   On macOS 10.5 systems, the SSH server daemon `sshd` allows password authentication by default. However, on macOS 10.6 systems, the SSH server daemon `sshd` disallows password authentication by default. .. Edit the configuration file `/etc/sshd_config` to set the `PasswordAuthentication` option to `yes`. .. Stop the SSH server daemon `sshd`.

   +

   ```
   $ sudo launchctl stop com.openssh.sshd
   ```

5. Start the SSH server daemon `sshd`.

   ```
   $ sudo launchctl start com.openssh.sshd
   ```

Next Steps

After you have completed the setup of SSH on a host, test the setup on the host as explained in testing-the-ssh-setup-on-a-host.

To Set up SSH on Linux Systems

1. Ensure that the following options in the configuration file `/etc/ssh/sshd_config` are set to `yes`:
   - `StrictModes`
   - `PubkeyAuthentication`
2. Determine if the SSH server daemon `sshd` is running.

   ```
   $ /sbin/service sshd status
   ```

3. If the SSH server daemon `sshd` is not running, start this daemon.

   If the daemon is running, no further action is required.

```
$ /sbin/service sshd start
```

This example confirms that the SSH server daemon `sshd` is running on a Linux system.

```
$ /sbin/service sshd status
openssh-daemon (pid  2373) is running...
```

Next Steps

After you have completed the setup of SSH on a host, test the setup on the host as explained in testing-the-ssh-setup-on-a-host.

Testing the SSH Setup on a Host

After setting up SSH on a host, test the setup to ensure that you can use SSH to contact the host from another host. Testing the SSH setup on a host verifies that the SSH server daemon `sshd` is running and that the SSH user has a valid user account on the host.

If you cannot use SSH to contact the host, troubleshoot the SSH setup before setting up SSH user authentication.

To Test the SSH Setup on a Host

1. From another host, use SSH to log in into the host that you are testing as the SSH user.

```
$ ssh -l user-name host-name
```

   user-name: The username for the SSH user's account on the host. **host-name**: The host name of the host that you are logging in to.

2. In response to the prompt, type your password.

   If this step succeeds, your setup of SSH is complete.

The first time that you connect to a host, you might be warned that the authenticity cannot be established and be asked if you want to continue connection. If you trust the host, answer `yes` to connect to the host.

Troubleshooting

To obtain diagnostic information, use the `-v` option of the command-line SSH client and the `-d` option of the SSH server daemon `sshd`. How to start the SSH server daemon `sshd` manually depends on the operating system and SSH provider that you are using.

If the SSH server daemon `sshd` is set up on a host that has a firewall, ensure that a rule is defined to allow inbound traffic on the SSH port. The default SSH port is port `22`.

If your connection is refused, the SSH server daemon `sshd` is not running, and you must start the daemon.

If your connection is accepted, but you cannot log in, ensure that the SSH user has a valid user account on the host.

Next Steps

After testing the SSH setup, set up SSH user authentication to enable SSH to authenticate users without prompting for a password. For more information, see setting-up-ssh-user-authentication.

Setting up SSH User Authentication

When a Payara Server subcommand uses SSH to log in to a remote host, Payara Server must be able to authenticate the SSH user. Setting up SSH user authentication ensures that this requirement is met.

Before setting up SSH user authentication, determine the authentication scheme to use. If SSH is already deployed at your site, the authentication scheme to use might already be chosen for you.

The following table lists the authentication schemes that Payara Server supports. The table also lists the advantages and disadvantages of each authentication scheme.

Authentication Scheme	Advantages	Disadvantages
Public key without encryption	Payara Server provides tools to simplify set up.	SSH must be configured to locate users' key files in the correct location. File access permissions for key files and the directory that contains the key files must be set correctly.
Public key with passphrase-protected encryption	This scheme is more secure than public key authentication without encryption.	SSH must be configured to locate users' key files in the correct location. File access permissions for key files and the directory that contains the key files must be set correctly. For each SSH user, Payara Server password aliases are required for the encryption passphrase.
Password	No SSH configuration is required to locate key files or to ensure that file access permissions are correct.	For each SSH user, Payara Server password aliases are required for the SSH password.

To Set up Public Key Authentication without Encryption

Use the `setup-ssh` subcommand in local mode to set up public key authentication without encryption. This subcommand enables you to set up public key authentication on multiple hosts in a single operation.

The `setup-ssh` subcommand generates a key pair and distributes the public key file to specified hosts. The private key file and the public key file are protected only by the file system's file access permissions.

If you require additional security, set up public key authentication with passphrase-protected encryption as explained in to-set-up-encrypted-public-key-authentication.

Before You Begin

Ensure that the following prerequisites are met:

- SSH is set up on each host where you are setting up public key authentication. For more information, see the following sections:
  - setting-up-cygwin-ssh-on-windows
  - setting-up-the-mks-toolkit-on-windows
  - setting-up-ssh-on-unix-and-linux-systems
- Only the SSH user has write access to the following files and directories on each host where you are setting up public key authentication:
  - The SSH user's home directory
  - The `~/.ssh` directory
  - The `authorized_key` file

If other users can write to these files and directories, the secure service might not trust the `authorized_key` file and might disallow public key authentication.

1. Generate an SSH key pair and distribute the public key file to the hosts where you are setting up public key authentication.

   NOTE | Only the options that are required to complete this task are provided in this step. For information about all the options for setting up an SSH key, see the setup-ssh help page.

   ```
   asadmin> setup-ssh [--sshuser sshuser] host-list
   ```

   sshuser: The SSH user for which you are generating the SSH key pair. If you are running the subcommand as the SSH user, you may omit this option. **host-list**: A space-separated list of the names of the hosts where the SSH public key is to be distributed.

   After generating the SSH key pair, the subcommand uses SSH to log in to each host in host-list as the SSH user to distribute the public key. Each time a password is

required to log in to a host, you are prompted for the SSH user's password.

2. In response to each prompt for a password, type the SSH user's password.

The following example generates and sets up an SSH key for the user `gfuser` on the hosts `sua01` and `sua02`. The command is run by the user `gfuser`.

```
asadmin> setup-ssh --generatekey=true sua01 sua02

Enter SSH password for gfuser@sua01>
Created directory /home/gfuser/.ssh
/usr/bin/ssh-keygen successfully generated the identification
/home/gfuser/.ssh/id_rsa
Copied keyfile /home/gfuser/.ssh/id_rsa.pub to gfuser@sua01
Successfully connected to gfuser@sua01 using keyfile /home/gfuser/.ssh/id_rsa
Copied keyfile /home/gfuser/.ssh/id_rsa.pub to gfuser@sua02
Successfully connected to gfuser@sua02 using keyfile /home/gfuser/.ssh/id_rsa
Command setup-ssh executed successfully.
```

Next Steps

After setting up public key authentication, test the setup by using `ssh` to log in as the SSH user to each host where the public key was distributed. For each host, log in first with the unqualified host name and then with the fully qualified name. If SSH does not prompt for password, public key authentication is set up correctly on the host.

If you are prompted for a password, verify that the public key file was copied correctly to the SSH user's `authorized_keys` file.

Troubleshooting

Setup might fail because file access permissions in the SSH user's home directory are too permissive. In this situation, ensure that the file access permissions in the SSH user's home directory meet the requirements for performing this procedure.

If you have set the file access permissions in the SSH user's home directory correctly, setup might still fail if you are using the MKS Toolkit. In this situation, correct the problem in one of the following ways:

- On each remote host, copy the public key file to the SSH user's `~/.ssh` directory and

import the file. To import the file, select the Secure Service tab in the MKS configuration GUI and click *Passwordless*.

- Disable strict modes.

See Also

- setting-up-cygwin-ssh-on-windows
- setting-up-the-mks-toolkit-on-windows
- setting-up-ssh-on-unix-and-linux-systems
- setup-ssh

You can also view the full syntax and options of the subcommand by typing `asadmin help setup-ssh` at the command line.

To Set up Encrypted Public Key Authentication

Encrypted key file authentication uses an encrypted private key file that is protected with a passphrase. This passphrase must be provided to use the private key to unlock the public key.

If you require encrypted public key authentication, you must use the SSH utility `ssh-keygen` to generate an SSH key pair with an encrypted private key. You can then use the `setup-ssh` subcommand to distribute the public key file to specified hosts.

To use the encrypted key file, Payara Server requires the passphrase with which the key file was encrypted. To provide this passphrase securely to Payara Server, create a Payara Server password alias to represent the passphrase and store this alias in a password file that is passed to the asadmin-1m utility.

NOTE	Only the options that are required to complete this task are provided in each step. For information about all the options for the commands and subcommands in this task, see their help pages or man pages.

Before You Begin

Ensure that the following prerequisites are met:

- SSH is set up on each host where you are setting up public key authentication. For more information, see the following sections:
  - setting-up-cygwin-ssh-on-windows
  - setting-up-the-mks-toolkit-on-windows
  - setting-up-ssh-on-unix-and-linux-systems
- Only the SSH user has write access to the following files and directories on each host where you are setting up public key authentication:
  - The SSH user's home directory
  - The `~/.ssh` directory
  - The `authorized_key` file

  If other users can write to these files and directories, the secure service might not trust the `authorized_key` file and might disallow public key authentication.

  1. Generate an SSH key pair with an encrypted private key file.

     Use the SSH utility `ssh-keygen` for this purpose.

     ```
     $ ssh-keygen -t type
     ```

     type: The algorithm that is to be used for the key and which must be `rsa`, `dsa`, or `rsa1`.

     The `ssh-keygen` utility prompts you for a file in which to save the key.

  2. To simplify the distribution of the key file, accept the default file.

     The `ssh-keygen` utility prompts you for a passphrase.

  3. In response to the prompt, type your choice of passphrase for encrypting the private key file. The `ssh-keygen` utility prompts you to type the passphrase

again.

4. In response to the prompt, type the passphrase that you set in Step **3**.
5. Distribute the public key file to the hosts where you are setting up public key authentication.

   Use the "`setup-ssh`" `asadmin` subcommand for this purpose.

```
$ asadmin setup-ssh --generatekey=false host-list
```

host-list: A space-separated list of the names of the hosts where the SSH public key is to be distributed.

+ The subcommand uses SSH to log in to each host in host-list as the SSH user to distribute the public key. Each time a passphrase or a password is required to log in to a host, you are prompted for the passphrase or the SSH user's password.

1. In response to each prompt, type the requested information.
   - In response to each prompt for a passphrase, type the passphrase that you set in Step **3**.
   - In response to each prompt for a password, type the SSH user's password.
2. Create a Payara Server password alias for the passphrase that you set in Step **3**.
   a. Ensure that the DAS is running.

      Remote subcommands require a running server.

   b. Run the "`create-password-alias`" `asadmin` subcommand.

      ```
      $ asadmin create-password-alias alias-name
      ```

      alias-name: Your choice of name for the alias that you are creating.

      The `create-password-alias` subcommand prompts you to type the passphrase for which you are creating an alias.

c. In response to the prompt, type the passphrase that you set in Step **3**.

   The `create-password-alias` subcommand prompts you to type the passphrase again.

d. In response to the prompt, type the passphrase that you set in Step **3** again.

3. Create a plain text file that contains the following entry for the passphrase alias:

```
AS_ADMIN_SSHKEYPASSPHRASE=${ALIAS=alias-name}
```

NOTE

When you create an `SSH` node, pass this file as the `--passwordfile` option of the `asadmin` utility. For more information, see to-create-an-ssh-node.

The following example generates an SSH key pair with an encrypted private key for the user `gfadmin` and distributes the public key to the hosts `sj01` and `ja02`. The example also creates an alias that is named `ssh-key-passphrase` for the private key's passphrase.

```
$ ssh-keygen -t rsa

Generating public/private rsa key pair.
Enter file in which to save the key (/home/gfadmin/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/gfadmin/.ssh/id_rsa.
Your public key has been saved in /home/gfadmin/.ssh/id_rsa.pub.
The key fingerprint is:
db:b5:f6:0d:fe:16:33:91:20:64:90:1a:84:66:f5:d0 gfadmin@dashost
$ asadmin setup-ssh --generatekey=false sj01 sj02
Key /home/gfadmin/.ssh/id_rsa is encrypted
Enter key passphrase>
Enter SSH password for gfadmin@sj01>
Copied keyfile /home/gfadmin/.ssh/id_rsa.pub to gfadmin@sj01
Successfully connected to gfadmin@sj01 using keyfile /home/gfadmin/.ssh/id_rsa
Successfully connected to gfadmin@sj02 using keyfile /home/gfadmin/.ssh/id_rsa
SSH public key authentication is already configured for gfadmin@sj02
Command setup-ssh executed successfully.
$ asadmin create-password-alias ssh-key-passphrase
Enter the alias password>
Enter the alias password again>
Command create-password-alias executed successfully.
```

The entry in the password file for the `ssh-key-passphrase` alias is as follows:

```
AS_ADMIN_SSHKEYPASSPHRASE=${ALIAS=ssh-key-passphrase}
```

Troubleshooting

Setup might fail because file access permissions in the SSH user's home directory are too permissive. In this situation, ensure that the file access permissions in the SSH user's home directory meet the requirements for performing this procedure.

If you have set the file access permissions in the SSH user's home directory correctly, setup might still fail if you are using the MKS Toolkit. In this situation, correct the problem in one of the following ways:

- On each remote host, copy the public key file to the SSH user's `~/.ssh` directory and import the file. To import the file, select the Secure Service tab in the MKS configuration GUI and click *Passwordless*.
- Disable strict modes.
- setting-up-cygwin-ssh-on-windows
- setting-up-the-mks-toolkit-on-windows
- setting-up-ssh-on-unix-and-linux-systems
- asadmin-1m
- create-password-alias
- setup-ssh
- `ssh-keygen`(1)

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help create-password-alias`
- `asadmin help setup-ssh`

To Set up Password Authentication

To use SSH to log in to a remote host, Payara Server requires the SSH user's password. To provide this password securely to Payara Server, create a Payara Server password alias to represent the password and store this alias in a password file that is passed to the asadmin-1m utility.

Before You Begin

Ensure that SSH is set up on each host where you are setting up password authentication.

1. Ensure that the DAS is running.

   Remote subcommands require a running server.

2. Create an alias for the SSH user's password.

NOTE	Only the options that are required to complete this task are provided in this step. For information about all the options for creating a password alias, see the create-password-alias help page.

```
asadmin> create-password-alias alias-name
```

alias-name: Your choice of name for the alias that you are creating.

+ The `create-password-alias` subcommand prompts you to type the password for which you are creating an alias.

1. In response to the prompt, type the SSH user's password.

   The `create-password-alias` subcommand prompts you to type the password again.

2. In response to the prompt, type the SSH user's password again.
3. Create a plain text file that contains the following entry for the password alias:

```
AS_ADMIN_SSHPASSWORD=${ALIAS=alias-name}
```

alias-name: The alias name that you specified in Step 2.

NOTE	When you create an `SSH` node, pass this file as the `--passwordfile` option of the `asadmin` utility. For more information, see to-create-an-ssh-node.

The following example creates an alias that is named `ssh-password` for the SSH user's password.

```
$ asadmin create-password-alias ssh-password
Enter the alias password>
Enter the alias password again>
Command create-password-alias executed successfully.
```

The entry in the password file for the `ssh-password` alias is as follows:

```
AS_ADMIN_SSHPASSWORD=${ALIAS=ssh-password}
```

See Also

- setting-up-cygwin-ssh-on-windows
- setting-up-the-mks-toolkit-on-windows
- setting-up-ssh-on-unix-and-linux-systems
- "`asadmin`(1M)"
- create-password-alias

You can also view the full syntax and options of the subcommand by typing `asadmin help create-password-alias` at the command line.

Installing and Removing Payara Server Software on Multiple Hosts

Payara Server software must be installed on all hosts where Payara Server will run. How to install Payara Server software on multiple hosts depends on the degree of control that you require over the installation on each host.

- If you require complete control over the installation on each host, install the software from a Payara Server distribution on each host individually.
- If the same setup on each host is acceptable, copy an existing Payara Server installation to the hosts. For more information, see to-copy-a-payara-server-installation-to-multiple-hosts.

Payara Server also enables you to remove Payara Server software from multiple hosts in a single operation. For more information, see to-remove-payara-server-software-from-multiple-hosts.

To Copy a Payara Server Installation to Multiple Hosts

Use the `install-node-ssh` subcommand in local mode to copy an installation of Payara Server software to multiple hosts.

Before You Begin

Ensure that SSH is set up on the host where you are running the subcommand and on each host where you are copying the Payara Server software.

Run the appropriate subcommand for the protocol that is set up for communication between the hosts.

- If SSH is set up for communication between the hosts, run the `install-node-ssh` subcommand.

NOTE	Only the options that are required to complete this task are provided in this step. For information about all the options for copying an installation of Payara Server software, see the link install-node-ssh help page.

  ```
  asadmin> install-node-ssh host-list
  ```

  host-list: A space-separated list of the names of the hosts where you are copying the installation of Payara Server software.

The following example copies the Payara Server software on the host where the subcommand is run to the default location on the SSH-enabled hosts `sj03.example.com` and `sj04.example.com`.

```
asadmin> install-node-ssh sj03.example.com sj04.example.com

Created installation zip /home/gfuser/glassfish2339538623689073993.zip
Successfully connected to gfuser@sj03.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Copying /home/gfuser/payara2339538623689073993.zip (81395008 bytes) to
sj03.example.com:/export/payara
Installing payara2339538623689073993.zip into sj03.example.com:/export/payara
Removing sj03.example.com:/export/payara/payara2339538623689073993.zip
Fixing file permissions of all files under sj03.example.com:/export/payara/bin
Successfully connected to gfuser@sj04.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Copying /home/gfuser/payara2339538623689073993.zip (81395008 bytes) to
sj04.example.com:/export/payara
Installing payara2339538623689073993.zip into sj04.example.com:/export/payara
Removing sj04.example.com:/export/payara/payara2339538623689073993.zip
Fixing file permissions of all files under sj04.example.com:/export/payara/bin
Command install-node-ssh executed successfully
```

See Also

- install-node-ssh

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help install-node-ssh`

To Remove Payara Server Software from Multiple Hosts

Use the `uninstall-node-ssh` subcommand in local mode to remove Payara Server software from multiple hosts.

Before You Begin

Ensure that the following prerequisites are met:

- SSH is set up on the host where you are running the subcommand and on each host from which you are removing the Payara Server software.

- No process is accessing the parent of the base installation directory for the Payara Server software or any subdirectory of this directory.
- The parent of the base installation directory for the Payara Server software is the same on each host from which you are removing the Payara Server software.
- For hosts that use SSH for remote communication, the configuration of the following items is the same on each host:
  - SSH port
  - SSH user
  - SSH key file

Run the appropriate subcommand for the protocol that is set up for communication between the hosts.

- If SSH is set up for communication between the hosts, run the `uninstall-node-ssh` subcommand.

> **NOTE** Only the options that are required to complete this task are provided in this step. For information about all the options for removing Payara Server software, see the uninstall-node-ssh help page.

```
asadmin> uninstall-node-ssh host-list
```

host-list: A space-separated list of the names of the hosts from which you are removing Payara Server software.

The following example removes Payara Server software on the SSH-enabled hosts `sj03.example.com` and `sj04.example.com` from the default location.

```
asadmin> uninstall-node-ssh sj03 sj04
Successfully connected to gfuser@sj03.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Successfully connected to gfuser@sj04.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Command uninstall-node-ssh executed successfully.
```

See Also

- uninstall-node-ssh

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help uninstall-node-ssh`

Administering Payara Server Nodes

A node represents a host on which the Payara Server software is installed. A node must exist for every host on which Payara Server instances reside.

A node's configuration contains information about the host such as the name of the host and the location where the Payara Server is installed on the host.

The following topics are addressed here:

- "Types of Payara Server Nodes"
- "Creating, Listing, Testing, and Deleting SSH Nodes"
- "Creating, Listing, and Deleting CONFIG Nodes"
- "Creating, Listing, and Deleting DOCKER Nodes"
- "Updating and Changing the Type of a Node"

Types of Payara Server Nodes

Each Payara Server node is one of the following types of node:

`SSH`: An `SSH` node supports communication over secure shell (SSH). If SSH is set up, and you plan to administer your Payara Server instances centrally, the instances must reside on `SSH` nodes.

+ An `SSH` node's configuration contains the information that is required to connect to the host through SSH. This information includes, for example, the username of the SSH user and the port number for SSH connections to the host. **`CONFIG`**: A `CONFIG` node

does not support remote communication. If SSH is set up, and you plan to administer your instances locally, the instances can reside on `CONFIG` nodes. You cannot use `CONFIG` nodes for instances that you plan to administer centrally.

+ Each domain contains a predefined `CONFIG` node that is named `localhost-` `` `domain, where domain is the name of the domain. On the host where the domain administration server (DAS) is running, this node represents the local host. `` **`DOCKER`**: A Docker node is similar to the existing SSH and CONFIG nodes in that it is a virtual representation of a Payara Server installation, providing connection details that the Domain Administration Server (DAS) uses to communicate with the installation to create, stop, start, and delete instances.

+ Where it differs from these existing node types is that instances created against it are actually housed in Docker containers, and so also contains details on how to communicate with the Docker engine of the remote machine.

To Create an `SSH` Node

Use the `create-node-ssh` subcommand in remote mode to create an `SSH` node.

Before You Begin

Ensure that the SSH user can use SSH to log in to the host that the node will represent. By default, the `create-node-ssh` subcommand validates the node's parameters and the SSH connection to the host.

If the SSH user cannot use SSH to log in to the host, the validation fails.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the `create-node-ssh` subcommand.

NOTE	Only the options that are required to complete this task are provided in this step. For information about all the options for configuring the node, see the "`create-node-ssh`" help page.

If you are using password authentication for the SSH user, you must specify a

password file through the `--passwordfile` option of the "`asadmin`" utility. For more information about SSH user authentication, see link:Technical Documentation/Payara Server Documentation/High Availability/ssh-setup.adoc#setting-up-ssh-user-authentication[Setting Up SSH User Authentication].

```
asadmin> create-node-ssh --nodehost node-host [--installdir install-dir ] node-name
```

node-host: The name of the host that the node represents. The name of the host must be specified. Otherwise, an error occurs. **install-dir**: The full path to the parent of the base installation directory of the Payara Server software on the host, for example, `/export/payara/`. If the Payara Server software is installed in the same directory on the node's host and the DAS host, you can omit this option. **node-name**: Your choice of name for the node that you are creating.

The following example creates the `SSH` node `sj01` to represent the host `sj01.example.com`. The Payara Server software is installed in the same directory on the DAS host and on the host `sj01.example.com`.

```
asadmin> create-node-ssh --nodehost sj01.example.com sj01

Command create-node-ssh executed successfully.
```

Troubleshooting

The `create-node-ssh` subcommand might fail to create the node and report the error `Illegal sftp packet len`. If this error occurs, ensure that no the startup file on the remote host displays text for noninteractive shells. Examples of startup files are `.bashrc`, `.cshrc`, `.login`, and `.profile`.

The SSH session interprets any text message that is displayed during login as a file-transfer protocol packet. Therefore, any statement in a startup file that displays text messages corrupts the SSH session, causing this error.

See also

"`create-node-ssh`"

You can also view the full syntax and options of the subcommand by typing `asadmin help create-node-ssh` at the command line.

Next Steps

After creating a node, you can create instances on the node as explained in the following sections:

- to-create-an-instance-centrally
- to-create-an-instance-locally

To List `SSH` Nodes in a Domain

Use the `list-nodes-ssh` subcommand in remote mode to obtain information about existing `SSH` nodes in a domain.

NOTE	To obtain information about all existing nodes in a domain, use the "`list-nodes`" subcommand.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the "`list-nodes-ssh`" subcommand.

```
asadmin> list-nodes-ssh
```

This example lists the name, type, and host of all `SSH` nodes in the current domain.

```
asadmin> list-nodes-ssh
sj01  SSH  sj01.example.com
sj02  SSH  sj02.example.com
Command list-nodes-ssh executed successfully.
```

The following example lists detailed information about all `SSH` nodes in the current domain.

```
asadmin> list-nodes-ssh --long=true
```

```
NODE NAME    TYPE    NODE HOST          INSTALL DIRECTORY    REFERENCED BY
sj01         SSH     sj01.example.com   /export/payara   pmd-i1
sj02         SSH     sj02.example.com   /export/payara   pmd-i2
Command list-nodes-ssh executed successfully.
```

See also

- "`list-nodes`"
- "`list-nodes-ssh`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help list-nodes`
- `asadmin help list-nodes-ssh`

To Test If an `SSH` Node Is Reachable

Use the `ping-node-ssh` subcommand in remote mode to test if an `SSH` node is reachable.

Before You Begin

Ensure that SSH is configured on the host where the DAS is running and on the host that the node represents.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the `ping-node-ssh` subcommand.

NOTE	Only the options that are required to complete this task are provided in this step. For information about all the options for testing the node, see the "`ping-node-ssh`" help page.

```
asadmin> ping-node-ssh node-name
```

node-name: The name of the node to test.

The following example tests if the `SSH` node `sj01` is reachable.

```
asadmin> ping-node-ssh sj01
Successfully made SSH connection to node sj01 (sj01.example.com)
Command ping-node-ssh executed successfully.
```

See also

"`ping-node-ssh`"

You can also view the full syntax and options of the subcommand by typing `asadmin help ping-node-ssh` at the command line.

To Delete an `SSH` Node

Use the `delete-node-ssh` subcommand in remote mode to delete an `SSH` node.

Deleting a node removes the node from the configuration of the DAS. The node's directories and files are deleted when the last Payara Server instance that resides on the node is deleted.

Before You Begin

Ensure that no Payara Server instances reside on the node that you are deleting. For information about how to delete an instance, see the following sections.

- to-delete-an-instance-centrally
- to-delete-an-instance-locally

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Confirm that no instances reside on the node that you are deleting.

   ```
   asadmin> list-nodes-ssh --long=true
   ```

3. Run the "`delete-node-ssh`" subcommand.

   ```
   asadmin> delete-node-ssh node-name
   ```

node-name: The name of the node that you are deleting.

The following example confirms that no instances reside on the `SSH` node `sj01` and deletes the node `sj01`.

```
asadmin> list-nodes-ssh --long=true
NODE NAME    TYPE    NODE HOST          INSTALL DIRECTORY    REFERENCED BY
sj01         SSH     sj01.example.com   /export/payara
sj02         SSH     sj02.example.com   /export/payara   pmd-i2

Command list-nodes-ssh executed successfully.
asadmin> delete-node-ssh sj01
Command delete-node-ssh executed successfully.
```

See also

- to-delete-an-instance-centrally
- to-delete-an-instance-locally
- "`delete-node-ssh`"
- "`list-nodes-ssh`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help delete-node-ssh`
- `asadmin help list-nodes-ssh`

Creating, Listing, and Deleting `CONFIG` Nodes

A `CONFIG` node does not support remote communication. If SSH is set, up and you plan to administer your instances locally, the instances can reside on `CONFIG` nodes. You cannot use `CONFIG` nodes for instances that you plan to administer centrally.

Payara Server enables you to create `CONFIG` nodes for use by instances, obtain information about `CONFIG` nodes, and delete `CONFIG` nodes that are no longer required.

The following topics are addressed here:

- "To Create a CONFIG Node"
- "To List CONFIG Nodes in a Domain"
- "To Delete a CONFIG Node"

To Create a `CONFIG` Node

Use the `create-node-config` command in remote mode to create a `CONFIG` node.

NOTE

If you create an instance locally on a host for which no nodes are defined, you can create the instance without creating a node beforehand. In this situation, Payara Server creates a `CONFIG` node for you. The name of the node is the unqualified name of the host.
For more information, see to-create-an-instance-locally.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the `create-node-config` subcommand.

NOTE

Only the options that are required to complete this task are provided in this step. For information about all the options for configuring the node, see the "`create-node-config`" help page.

```
asadmin> create-node-config [--nodehost node-host] [--installdir install-dir ]
node-name
```

node-host: The name of the host that the node represents. You may omit this option. The name of the host can be determined when instances that reside on the node are created. **install-dir**: The full path to the parent of the base installation directory of the Payara Server software on the host, for example, `/export/payara/`. You may omit this option. The installation directory can be determined when instances that reside on the node are created. **node-name**: Your choice of name for the node that you are creating.

The following example creates the `CONFIG` node `cfg01`. The host that the node represents and the installation directory of the Payara Server software on the host are to

be determined when instances are added to the node.

```
asadmin> create-node-config cfg01
Command create-node-config executed successfully.
```

See also

"`create-node-config`"

You can also view the full syntax and options of the subcommand by typing `asadmin help create-node-config` at the command line.

Next Steps

After creating a node, you can create instances on the node as explained in to-create-an-instance-locally.

To List `CONFIG` Nodes in a Domain

Use the `list-nodes-config` subcommand in remote mode to obtain information about existing `CONFIG` nodes in a domain.

NOTE	To obtain information about all existing nodes in a domain, use the "`list-nodes`" subcommand.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the "`list-nodes-config`" subcommand.

```
asadmin> list-nodes-config
```

The following example lists the name, type, and host of all `CONFIG` nodes in the current domain.

```
asadmin> list-nodes-config
localhost-domain1  CONFIG  localhost
cfg01  CONFIG  cfg01.example.com
cfg02  CONFIG  cfg02.example.com
Command list-nodes-config executed successfully.
```

The following example lists detailed information about all `CONFIG` nodes in the current domain.

```
asadmin> list-nodes-config --long=true

NODE NAME           TYPE     NODE HOST            INSTALL DIRECTORY    REFERENCED
BY
localhost-domain1   CONFIG   localhost            /export/payara
cfg01               CONFIG   cfg01.example.com    /export/payara   yml-i1
cfg02               CONFIG   cfg02.example.com    /export/payara   yml-i2
Command list-nodes-config executed successfully.
```

See also

- "`list-nodes`"
- "`list-nodes-config`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help list-nodes`
- `asadmin help list-nodes-config`

To Delete a `CONFIG` Node

Use the `delete-node-config` subcommand in remote mode to delete a `CONFIG` node.

Deleting a node removes the node from the configuration of the DAS. The node's directories and files are deleted when the last Payara Server instance that resides on the node is deleted.

Before You Begin

Ensure that no Payara Server instances reside on the node that you are deleting. For information about how to delete an instance that resides on a `CONFIG` node, see to-delete-an-instance-locally.

1. Ensure that the DAS is running. Remote subcommands require a running server.

2. Confirm that no instances reside on the node that you are deleting.

```
asadmin> list-nodes-config --long=true
```

3. Run the "`delete-node-config`" subcommand.

```
asadmin> delete-node-config node-name
```

node-name: The name of the node that you are deleting.

The following example confirms that no instances reside on the `CONFIG` node `cfg01` and deletes the node `cfg01`.

```
asadmin> list-nodes-config --long=true

NODE NAME            TYPE     NODE HOST           INSTALL DIRECTORY    REFERENCED BY
localhost-domain1    CONFIG   localhost           /export/payara
cfg01                CONFIG   cfg01.example.com   /export/payara
cfg02                CONFIG   cfg02.example.com   /export/payara   yml-i2
Command list-nodes-config executed successfully.
asadmin> delete-node-config cfg01
Command delete-node-config executed successfully.
```

See also

- to-delete-an-instance-locally
- "`delete-node-config`"
- "`list-nodes-config`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help delete-node-config`
- `asadmin help list-nodes-config`

Creating, Listing and Deleting Docker Nodes

As previously explained, a Docker node shares similarities with existing SSH and CONFIG nodes as it serves as a virtual representation of a Payara Server installation.

IMPORTANT	The Docker engine must expose its REST admin service on an HTTP port that is reacheable by the DAS to configure a DOCKER node.

Exposing the Docker REST Admin Service

To expose the REST admin service of the Docker engine, you need to specify the `-H` option in your `DOCKER_OPTS` environment variable, or otherwise specify it in the command that starts the docker service.

For example:

```
DOCKER_OPTS="-H=0.0.0.0:2376"
```

You can test that it is exposed by navigating to it with your browser (assuming you're exposing the port in the example above): `http://localhost:2376/info`

Creating a Docker Node

Creating a Docker node is done in a similar manner to creating an SSH or CONFIG node, either by asadmin command, or by the Nodes page of the admin console.

Docker nodes do have some unique configuration properties however:

- **Docker Password File** - This is the fully-qualified path of the password file that the Docker instance will use for authentication against the DAS.

  Please note, that this path should be the path to the file on the **remote** machine. This file should be a standard Payara Server password file as you would use with asadmin. This must be specified, as Docker instances require secure admin to be enabled to start.

- TLS - Whether or not to use HTTPS to communicate with the Docker engine or not.
- Docker Port - The port that the Docker engine is listening on. Defaults to `2376`.
- Docker Image - The Docker image to use. Payara Server will default to using the `payara/server-node:7.2.0` image.

The configuration options of CONFIG nodes are also available, namely `nodehost`, `nodedir`, and `installdir`. Specifying the `nodehost` option remains mandatory, but for docker nodes the `installdir` and `nodedir` options can safely be left as their defaults unless you're specifying your own Docker image.

Please take particular note of the *Docker Password File* option: this is mandatory and **must be present on the remote machine**. Payara Server does not currently support propagating password files from the DAS to your remote machines. Secure admin must also be enabled for Docker Nodes to function correctly, which entails setting an admin password. See below for an example of enabling secure admin for the default *admin* user:

```
asadmin change-admin-password --user admin
Enter the admin password>
Enter the new admin password>
Enter the new admin password again>
Command change-admin-password executed successfully.

asadmin enable-secure-admin
Authentication failed with password from login store
Enter admin password for user "admin">
You must restart all running servers for the change in secure admin to take effect.
Command enable-secure-admin executed successfully.

asadmin restart-domain
```

As an example of an extremely simple password file, here is the contents of a sample, `passwordfile.txt`:

```
AS_ADMIN_PASSWORD=changeit
```

Where `changeit` is the password of the default *admin* user.

Once you have ensured that you have secure admin enabled, and that a password file granting access to the DAS is present on the remote machine, you can run the asadmin command for creating a Docker node:

```
asadmin create-node-docker --nodehost localhost --dockerPasswordFile
/opt/passwordfile.txt --dockerport 2376 docker-node-1
```

TIP	The admin console page for creating Docker nodes is the same as creating the other node types: simply select `DOCKER` as the node type from the dropdown.

Configuring TLS

As Docker must expose an HTTP port, it is recommended that it be protected using TLS outside of development environments.

A full guide for how to do so can be found here In short, create self-signed TLS certificates (or use your own signed ones), and add them to your `DOCKER_OPTS` variable to secure the Docker REST admin service:

```
DOCKER_OPTS="--tlsverify --tlscacert=/home/anon/tls/ca.pem
--tlscert=/home/anon/tls/server-cert.pem --tlskey=/home/anon/tls/server-key.pem
-H=0.0.0.0:2376"
```

Assuming you followed this guide, you will need to perform the following steps to allow Payara Server to communicate with the Docker engine correctly:

1. Import the CA certificate into the Domain truststore (defaults to `domain-dir/config/cacerts.p12`)
2. Create a PKCS12 bundle from the client certificate and key
3. Import the generated PKCS12 bundle into the Domain's keystore (defaults to `domain-dir/config/keystore.p12`)

Once you've enabled TLS for the Docker REST admin service and added the necessary certificates & keys to Payara Server, you can create a Docker Node with TLS enabled like so:

```
asadmin create-node-docker --nodehost localhost --useTls true --dockerPasswordFile
/opt/passwordfile.txt --dockerport 2376 docker-node-1
```

IMPORTANT	Please ensure that the `useTLS` option of the node matches whether or not you have configured TLS for the Docker engine, as

> otherwise any communication with the Docker engine will fail.

To Delete a DOCKER Node

Use the `delete-node-docker` subcommand in remote mode to delete a DOCKER node.

Deleting a node removes the node from the configuration of the DAS and the Docker container that corresponds to the node's system is stopped and removed. Follow these steps to remove a Docker node:

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Confirm that no instances reside on the node that you are deleting.

```
asadmin list-nodes-config --long=true
```

3. Run the "`delete-node-docker`" subcommand.

```
asadmin> delete-node-docker node-name
```

node-name: The name of the node that you are deleting.

Updating and Changing the Type of a Node

Payara Server enables you to update the configuration data of any node and to change the type of a node.

To Update an `SSH` Node

Use the `update-node-ssh` subcommand in remote mode to update an `SSH` node.

Options of this subcommand specify the new values of the node's configuration data. If you omit an option, the existing value is unchanged.

Before You Begin

Ensure that the following prerequisites are met:

- SSH is configured on the host where the DAS is running and on the host that the node represents.
- The node that you are updating exists.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the `update-node-ssh` subcommand.

```
asadmin> update-node-ssh options node-name
```

options: Options of the `update-node-ssh` subcommand for changing the node's configuration data. For information about these options, see the "`update-node-ssh`" help page. **node-name**: The name of the `SSH` node to update.

The following example updates the host that the node `sj01` represents to `adc01.example.com`.

```
asadmin> update-node-ssh --nodehost adc01.example.com sj01
Command update-node-ssh executed successfully.
```

See also

"`update-node-ssh`"

You can also view the full syntax and options of the subcommand by typing `asadmin help update-node-ssh` at the command line.

To Update a `CONFIG` Node

Use the `update-node-config` subcommand in remote mode to update a `CONFIG` node.

Options of this subcommand specify the new values of the node's configuration data. If you omit an option, the existing value is unchanged.

Before You Begin

Ensure that the node that you are updating exists.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the `update-node-config` subcommand.

```
asadmin> update-node-config options node-name
```

options: Options of the `update-node-config` subcommand for changing the node's configuration data. For information about these options, see the "`update-node-config`" help page. **node-name**: The name of the `CONFIG` node to update.

The following example updates the host that the node `cfg02` represents to `adc02.example.com`.

```
asadmin> update-node-config --nodehost adc02.example.com cfg02
Command update-node-config executed successfully.
```

See also

"`update-node-config`"

You can also view the full syntax and options of the subcommand by typing `asadmin help update-node-config` at the command line.

To Change the Type of a Node

The subcommands for updating a node can also be used to change the type of a node.

IMPORTANT	Due to the structure and mechanisms used to configure `DOCKER` nodes, it is not possible to switch the type of a `DOCKER` node to either `CONFIG`/`SSH` nodes or viceversa.

Changing the type of a `CONFIG` node to a `SSH` node enables remote communication for the node. It is also possible to change an `SSH` node to a `CONFIG` node too.

As part of the process of changing the type of a node, you can also change other configuration data for the node.

Options of the subcommands for updating a node specify the new values of the node's configuration data. For most options, if you omit the option, the existing value is unchanged. However, default values are applied in the following situations:

- Any of the following options of the `update-node-ssh` subcommand is omitted:
  - `--sshport`
  - `--sshuser`
  - `--sshkeyfile`

CAUTION	Changing an `SSH` node to a `CONFIG` node disables remote communication for the node.

Before You Begin

Ensure that the following prerequisites are met:

- SSH is configured on the host where the DAS is running and on the host that the node represents.
- The node the type of which you are changing exists.

Execute the following steps:

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the appropriate subcommand for updating a node, depending on the type of the node after the change.

   config-or-ssh-node-name: The name of the `CONFIG` node or the `SSH` node to change. * To change the type of a node to `SSH`, run the `update-node-ssh` subcommand on the node.

   + options: Options of the `update-node-ssh` subcommand for changing the node's configuration data. For information about these options, see the "`update-node-ssh`" help page.

- To change the type of a node to `CONFIG`, run the `update-node-config` subcommand on the node.

```
asadmin> update-node-config [options] ssh-node-name
```

options: Options of the `update-node-config` subcommand for changing the node's configuration data. For information about these options, see the "`update-node-config`" help page. **ssh-node-name**: The name of the `SSH` node to change.

The following example changes the `CONFIG` node `cfg02` to an `SSH` node.

```
asadmin> update-node-ssh cfg02
Command update-node-ssh executed successfully.
```

See also

- "`update-node-config`"
- "`update-node-ssh`"

You can also view the full syntax and options of the subcommand by typing the following commands at the command line.

- `asadmin help update-node-config`
- `asadmin help update-node-ssh`

Administering Payara Server Clusters

IMPORTANT	Clusters have been deprecated and officially replaced with Deployment Groups. We do not recommend setting up highly-available architectures using traditional clusters.

A cluster is a collection of Payara Server instances that work together as one logical entity. A cluster provides a runtime environment for one or more Jakarta Platform, Enterprise Edition (Jakarta EE) applications. A cluster provides high availability through

failure protection, scalability, and load balancing.

About Payara Server Clusters

A cluster is a named collection of Payara Server instances that share the same applications, resources, and configuration information.

For information about Payara Server instances, see administering-payara-server-instances.

Payara Server enables you to administer all the instances in a cluster as a single unit from a single host, regardless of whether the instances reside on the same host or different hosts. You can perform the same operations on a cluster that you can perform on an un-clustered instance, for example, deploying applications and creating resources.

A cluster provides high availability through failure protection, scalability, and load balancing.

- Failure protection. If an instance or a host in a cluster fails, Payara Server detects the failure and recovers the user session state.

  If a load balancer is configured for the cluster, the load balancer redirects requests from the failed instance to other instances in the cluster.

  Because the same applications and resources are on all instances in the cluster, an instance can fail over to any other instance in the cluster.

  To enable the user session state to be recovered, each instance in a cluster sends in-memory state data to another instance. As state data is updated in any instance, the data is replicated.

- Scalability. If increased capacity is required, you can add instances to a cluster with no disruption in service. When an instance is added or removed, the changes are handled automatically.

- Load balancing. If instances in a cluster are distributed among different hosts, the workload can be distributed among the hosts to increase overall system throughput.

Group Management Service

The Group Management Service (GMS) is an infrastructure component that is enabled for the instances in a cluster. When GMS is enabled, if a clustered instance fails, the cluster and the Domain Administration Server (DAS) are aware of the failure and can take action when failure occurs.

NOTE	GMS is a core service of the Shoal framework.

About GMS in Payara Server

In previous versions of Payara Server, specifically Payara Server 4.x, clusters were implemented using the Group Management Service (GMS) and the Shoal framework to allow server instances to communicate between each other and coordinate high-availability features. GMS enables instances to participate in a cluster by detecting changes in cluster membership and notifying instances of the changes.

Starting in Payara Server 5, GMS has been replaced internally with the use of Hazelcast, the technology used to provision the server's Data Grid functionality. This replacement has been implemented in an encapsulated manner: Users will still interact with GMS settings via the Admin Console and Asadmin CLI, but under the covers the communication is managed through a set of adapters that use Hazelcast. This was done to keep a certain degree of backward compatibility between Payara Server 4 and 5.

To this effect, most of the configuration settings listed in the next section may not properly work as clusters have been effectively replaced by deployment groups.

GMS Configuration Settings

Payara Server has the following types of GMS settings:

- GMS cluster settings — These are determined during cluster creation. For more information about these settings, see to-create-a-cluster.

- GMS configuration settings — These are determined during configuration creation and are explained here.

The following GMS configuration settings are used in GMS for group discovery and failure detection:

`group-discovery-timeout-in-millis`: Indicates the amount of time (in milliseconds) an instance's GMS module will wait during instance startup for discovering other members of the group.

+ The `group-discovery-timeout-in-millis` timeout value should be set to the default or higher. The default is 5000.

`max-missed-heartbeats`: Indicates the maximum number of missed heartbeats that the health monitor counts before the instance can be marked as a suspected failure. GMS also tries to make a peer-to-peer connection with the suspected member. If the maximum number of missed heartbeats is exceeded and peer-to-peer connection fails, the member is marked as a suspected failure. The default value is `3`.

`heartbeat-frequency-in-millis`: Indicates the frequency (in milliseconds) at which a heartbeat is sent by each server instance to the cluster.

+ The failure detection interval is the `max-missed-heartbeats` multiplied by the `heartbeat-frequency-in-millis`. Therefore, the combination of defaults, 3 multiplied by 2000 milliseconds, results in a failure detection interval of 6 seconds.

+ Lowering the value of `heartbeat-frequency-in-millis` below the default would result in more frequent heartbeat messages being sent out from each member. This could potentially result in more heartbeat messages in the network than a system needs for triggering failure detection protocols. The effect of this varies depending on how quickly the deployment environment needs to have failure detection performed. That is, the (lower) number of retries with a lower heartbeat interval would make it quicker to detect failures.

+ However, lowering this value could result in false positives because you could

potentially detect a member as failed when, in fact, the member's heartbeat is reflecting the network load from other parts of the server. Conversely, a higher timeout interval results in fewer heartbeats in the system because the time interval between heartbeats is longer. As a result, failure detection would take a longer. In addition, a startup by a failed member during this time results in a new join notification but no failure notification, because failure detection and verification were not completed.

+ The default value is `2000`.

`verify-failure-waittime-in-millis`: Indicates the verify suspect protocol's timeout used by the health monitor. After a member is marked as suspect based on missed heartbeats and a failed peer-to-peer connection check, the verify suspect protocol is activated and waits for the specified timeout to check for any further health state messages received in that time, and to see if a peer-to-peer connection can be made with the suspect member. If not, then the member is marked as failed and a failure notification is sent.

+ The default value is `1500`.

`verify-failure-connect-timeout-in-millis`: Indicates the time it takes for the GMS to detect a hardware or network failure of a server instance. Be careful not to set this value too low. The smaller this timeout value is, the greater the chance of detecting false failures. That is, the instance has not failed but doesn't respond within the short window of time.

+ The default value is `10000`.

The heartbeat frequency, maximum missed heartbeats, peer-to-peer connection-based failure detection, and the verify timeouts are all needed to ensure that failure detection is robust and reliable in Payara Server.

For the dotted names for each of these GMS configuration settings, see dotted-names-for-gms-settings. For the steps to specify these settings, see to-preconfigure-nondefault-gms-configuration-settings.

Dotted Names for GMS Settings

Below are sample "`get`" subcommands to get all the GMS configuration settings (attributes associated with the referenced `mycfg` configuration) and GMS cluster settings (attributes and properties associated with a cluster named `mycluster`).

```
asadmin> get "configs.config.mycfg.group-management-service.*"

configs.config.mycfg.group-management-service.failure-detection.heartbeat-
frequency-in-millis=2000
configs.config.mycfg.group-management-service.failure-detection.max-missed-
heartbeats=3
configs.config.mycfg.group-management-service.failure-detection.verify-failure-
connect-timeout-in-millis=10000
configs.config.mycfg.group-management-service.failure-detection.verify-failure-
waittime-in-millis=1500
configs.config.mycfg.group-management-service.group-discovery-timeout-in-millis
=5000

asadmin> get clusters.cluster.mycluster

clusters.cluster.mycluster.config-ref=mycfg
clusters.cluster.mycluster.gms-bind-interface-address=${GMS-BIND-INTERFACE-ADDRESS
-mycluster}
clusters.cluster.mycluster.gms-enabled=true
clusters.cluster.mycluster.gms-multicast-address=228.9.245.47
clusters.cluster.mycluster.gms-multicast-port=9833
clusters.cluster.mycluster.name=mycluster

asadmin> get "clusters.cluster.mycluster.property.*"
clusters.cluster.mycluster.property.GMS_LISTENER_PORT=${GMS_LISTENER_PORT-
mycluster}
clusters.cluster.mycluster.property.GMS_MULTICAST_TIME_TO_LIVE=4
clusters.cluster.mycluster.property.GMS_LOOPBACK=false
clusters.cluster.mycluster.property.GMS_TCPSTARTPORT=9090
clusters.cluster.mycluster.property.GMS_TCPENDPORT=9200
```

The last `get` subcommand displays only the properties that have been explicitly set.

For the steps to specify these settings, see to-preconfigure-nondefault-gms-configuration-settings and to-change-gms-settings-after-cluster-creation.

To Preconfigure Nondefault GMS Configuration Settings

You can preconfigure GMS with values different than the defaults without requiring a restart of the DAS and the cluster.

1. Create a configuration using the "`copy-config`" subcommand. For example:

```
asadmin> copy-config default-config mycfg
```

For more information, see to-create-a-named-configuration.

2. Set the values for the new configuration's GMS configuration settings. For example:

```
asadmin> set configs.config.mycfg.group-management-service.group-discovery-
timeout-in-millis=8000
asadmin> set configs.config.mycfg.group-management-service.failure-
detection.max-missed-heartbeats=5
```

For a complete list of the dotted names for these settings, see dotted-names-for-gms-settings.

3. Create the cluster so it uses the previously created configuration. For example:

```
asadmin> create-cluster --config mycfg mycluster
```

You can also set GMS cluster settings during this step. For more information, see to-create-a-cluster.

4. Create server instances for the cluster. For example:

```
asadmin> create-instance --node localhost-domain1 --cluster mycluster instance01
asadmin> create-instance --node localhost-domain1 --cluster mycluster instance02
```

5. Start the cluster. For example:

```
asadmin> start-cluster mycluster
```

You can also view the full syntax and options of a subcommand by typing `asadmin help` subcommand at the command line.

To Change GMS Settings after Cluster Creation

To avoid the need to restart the DAS and the cluster, configure GMS configuration settings before cluster creation as explained in to-preconfigure-nondefault-gms-configuration-settings.

To avoid the need to restart the DAS and the cluster, configure the GMS cluster settings during cluster creation as explained in to-create-a-cluster.

Changing any GMS settings using the `set` subcommand after cluster creation requires a domain administration server (DAS) and cluster restart as explained here.

1. Ensure that the DAS and cluster are running. Remote subcommands require a running server.
2. Use the "`get`" subcommand to determine the settings to change. For example:

```
asadmin> get "configs.config.mycfg.group-management-service.*"

configs.config.mycfg.group-management-service.failure-detection.heartbeat-
frequency-in-millis=2000
configs.config.mycfg.group-management-service.failure-detection.max-missed-
heartbeats=3
configs.config.mycfg.group-management-service.failure-detection.verify-failure-
connect-timeout-in-millis=10000
configs.config.mycfg.group-management-service.failure-detection.verify-failure-
waittime-in-millis=1500
configs.config.mycfg.group-management-service.group-discovery-timeout-in-millis
=5000
```

   For a complete list of the dotted names for these settings, see dotted-names-for-gms-settings.

3. Use the "`set`" subcommand to change the settings. For example:

```
asadmin> set configs.config.mycfg.group-management-service.group-discovery-
timeout-in-millis=6000
```

4. Use the `get` subcommand again to confirm that the changes were made. For example:

```
asadmin> get configs.config.mycfg.group-management-service.group-discovery-
timeout-in-millis
```

5. Restart the DAS. For example:

```
asadmin> stop-domain domain1
asadmin> start-domain domain1
```

6. Restart the cluster. For example:

```
asadmin> stop-cluster mycluster
asadmin> start-cluster mycluster
```

You can also view the full syntax and options of a subcommand by typing `asadmin help` subcommand at the command line.

To Check the Health of Instances in a Cluster

The `list-instances` is the quickest way to evaluate the health of a cluster and to detect if cluster is properly operating; that is, all members of the cluster are running and visible to DAS.

1. Ensure that the DAS and cluster are running. Remote subcommands require a running server.
2. Check whether server instances in a cluster are running by using the "`list-instances`" subcommand.

This example checks the health of a cluster named `mycluster`.

```
asadmin> list-instances mycluster
instance01   running
instance02   running
Command list-instances executed successfully.
```

You can also view the full syntax and options of the subcommand by typing `asadmin help list-instances` at the command line.

Using the Multi-Homing Feature with GMS

Multi-homing enables Payara Server clusters to be used in an environment that uses multiple Network Interface Cards (NICs). A multi-homed host has multiple network connections, of which the connections may or may not be the on same network. Multi-homing provides the following benefits:

- Provides redundant network connections within the same subnet. Having multiple NICs ensures that one or more network connections are available for

communication.

- Supports communication across two or more different subnets. The DAS and all server instances in the same cluster must be on the same subnet for GMS communication, however.
- Binds to a specific IPv4 address and receives GMS messages in a system that has multiple IP addresses configured. The responses for GMS messages received on a particular interface will also go out through that interface.
- Supports separation of external and internal traffic.

Traffic Separation Using Multi-Homing

You can separate the internal traffic resulting from GMS from the external traffic. Traffic separation enables you plan a network better and augment certain parts of the network, as required.

Consider a simple cluster, `c1`, with three instances, `i101`, `i102`, and `i103`. Each instance runs on a different machine.

In order to separate the traffic, the multi-homed machine should have at least two IP addresses belonging to different networks. The first IP as the external IP and the second one as internal IP. The objective is to expose the external IP to user requests, so that all the traffic from the user requests would be through them. The internal IP is used only by the cluster instances for internal communication through GMS.

The following procedure describes how to set up traffic separation.

To configure multi-homed machines for GMS without traffic separation, skip the steps or commands that configure the `EXTERNAL-ADDR` system property, but perform the others.

To avoid having to restart the DAS or cluster, perform the following steps in the specified order.

To set up traffic separation, follow these steps:

---

1. Create the system properties `EXTERNAL-ADDR` and `GMS-BIND-INTERFACE-ADDRESS-c1` for the DAS.
   - `asadmin create-system-properties target server EXTERNAL-ADDR=192.155.35.4`
   - `asadmin create-system-properties target server GMS-BIND-INTERFACE-ADDRESS-c1=10.12.152.20`
2. Create the cluster with the default settings. Use the following command:

   ```
   asadmin create-cluster c1
   ```

   A reference to a system property for GMS traffic is already set up by default in the `gms-bind-interface-address` cluster setting. The default value of this setting is `${GMS-BIND-INTERFACE-ADDRESS-`cluster-name}`.
3. When creating the clustered instances, configure the external and GMS IP addresses. Use the following commands:
   - `asadmin create-instance node localhost cluster c1 systemproperties EXTERNAL-ADDR=192.155.35.5:GMS-BIND-INTERFACE-ADDRESS-c1=10.12.152.30 i101`
   - `asadmin create-instance node localhost cluster c1 systemproperties EXTERNAL-ADDR=192.155.35.6:GMS-BIND-INTERFACE-ADDRESS-c1=10.12.152.40 i102`
   - `asadmin create-instance node localhost cluster c1 systemproperties EXTERNAL-ADDR=192.155.35.7:GMS-BIND-INTERFACE-ADDRESS-c1=10.12.152.50 i103`
4. Set the address attribute of HTTP listeners to refer to the `EXTERNAL-ADDR` system properties. Use the following commands:

   ```
   asadmin set c1-config.network-config.network-listeners.network-listener.http-1.address=\${EXTERNAL-ADDR}
   asadmin set c1-config.network-config.network-listeners.network-listener.http-2.address=\${EXTERNAL-ADDR}
   ```

Creating, Listing, and Deleting Clusters

Payara Server enables you to create clusters, obtain information about clusters, and delete clusters that are no longer required.

To Create a Cluster

Use the `create-cluster` subcommand in remote mode to create a cluster.

To ensure that the GMS can detect changes in cluster membership, a cluster's GMS settings must be configured correctly. To avoid the need to restart the DAS and the cluster, configure a cluster's GMS settings when you create the cluster. If you change GMS settings for an existing cluster, the DAS and the cluster must be restarted to apply the changes.

When you create a cluster, Payara Server automatically creates a Message Queue cluster for the Payara Server cluster.

For more information about Message Queue clusters, see using-message-queue-broker-clusters-with-payara-server.

Before You Begin

If the cluster is to reference an existing named configuration, ensure that the configuration exists. For more information, see to-create-a-named-configuration.

If you are using a named configuration to preconfigure GMS settings, ensure that these settings have the required values in the named configuration.

For more information, see to-preconfigure-nondefault-gms-configuration-settings.

If you are configuring the cluster's GMS settings when you create the cluster, ensure that you have the following information:

- The address on which GMS listens for group events
- The port number of the communication port on which GMS listens for group events
- The maximum number of iterations or transmissions that a multicast message for

GMS events can experience before the message is discarded

- The lowest port number in the range of ports from which GMS selects a TCP port on which to listen
- The highest port number in the range of ports from which GMS selects a TCP port on which to listen

If the DAS is running on a multihome host, ensure that you have the Internet Protocol (IP) address of the network interface on the DAS host to which GMS binds.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the `create-cluster` subcommand.

NOTE	Only the options that are required to complete this task are provided in this step. For information about all the options for configuring the cluster, see the "`create-cluster`" help page.

- If multicast transport is available, run the `create-cluster` subcommand as follows:

```
asadmin> create-cluster --config configuration
--multicastaddress multicast-address --multicastport multicast-port
--properties GMS_MULTICAST_TIME_TO_LIVE=max-iterations:
GMS_TCPSTARTPORT=start-port:GMS_TCPENDPORT=end-port cluster-name
```

- If multicast transport is not available, run the `create-cluster` subcommand as follows:

```
asadmin> create-cluster --config configuration --properties
GMS_DISCOVERY_URI_LIST=discovery-instances:GMS_LISTENER_PORT=gms-port cluster-
name
```

configuration: An existing named configuration that the cluster is to reference. **multicast-address**: The address on which GMS listens for group events. **multicast-port**: The port number of the communication port on which GMS listens for group events. **max-iterations**: The maximum number of iterations or transmissions that a multicast

message for GMS events can experience before the message is discarded. **discovery-instances**: Instances to use for discovering the cluster. **gms-port**: The port number of the port on which the cluster listens for messages from GMS. **start-port**: The lowest port number in the range of ports from which GMS selects a TCP port on which to listen. The default is 9090. **end-port**: The highest port number in the range of ports from which GMS selects a TCP port on which to listen. The default value is 9200. **cluster-name**: Your choice of name for the cluster that you are creating. . If necessary, create a system property to represent the IP address of the network interface on the DAS host to which GMS binds. This step is necessary only if the DAS is running on a multihome host.

+

```
asadmin> create-system-properties
GMS-BIND-INTERFACE-ADDRESS-cluster-name=das-bind-address
```

cluster-name: The name that you assigned to the cluster in Step 2. **das-bind-address**: The IP address of the network interface on the DAS host to which GMS binds.

This example creates a cluster that is named `ltscluster` for which port 1169 is to be used for secure IIOP connections. Because the `--config` option is not specified, the cluster references a copy of the named configuration `default-config` that is named `ltscluster-config`.

```
asadmin> create-cluster --systemproperties IIOP_SSL_LISTENER_PORT=1169 ltscluster
Command create-cluster executed successfully.
```

Next Steps

After creating a cluster, you can add Payara Server instances to the cluster as explained in the following sections:

- to-create-an-instance-centrally
- to-create-an-instance-locally

See Also

- to-create-a-named-configuration
- to-preconfigure-nondefault-gms-configuration-settings
- using-message-queue-broker-clusters-with-payara-server
- "`create-cluster`"
- "`create-system-properties`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help create-cluster`
- `asadmin help create-system-properties`

To List All Clusters in a Domain

Use the `list-clusters` subcommand in remote mode to obtain information about existing clusters in a domain.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the "`list-clusters`" subcommand.

```
asadmin> list-clusters
```

This example lists all clusters in the current domain.

```
asadmin> list-clusters
pmdclust not running
ymlclust not running
Command list-clusters executed successfully.
```

This example lists the clusters that contain an instance that resides on the node `sj01`.

```
asadmin> list-clusters sj01
ymlclust not running
Command list-clusters executed successfully.
```

See Also

"`list-clusters`"

You can also view the full syntax and options of the subcommand by typing `asadmin help list-clusters` at the command line.

To Delete a Cluster

Use the `delete-cluster` subcommand in remote mode to remove a cluster from the DAS configuration.

If the cluster's named configuration was created automatically for the cluster and no other clusters or unclustered instances refer to the configuration, the configuration is deleted when the cluster is deleted.

Before You Begin

Ensure that following prerequisites are met:

- The cluster that you are deleting is stopped. For information about how to stop a cluster, see to-stop-a-cluster.
- The cluster that you are deleting contains no Payara Server instances. For information about how to remove instances from a cluster, see the following sections:
  - to-delete-an-instance-centrally
  - to-delete-an-instance-locally

Follow these steps:

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Confirm that the cluster is stopped.

```
asadmin> list-clusters cluster-name
```

cluster-name: The name of the cluster that you are deleting.

3. Confirm that the cluster contains no instances.

```
asadmin> list-instances cluster-name
```

cluster-name: The name of the cluster that you are deleting.

4. Run the "`delete-cluster`" subcommand.

```
asadmin> delete-cluster cluster-name
```

cluster-name: The name of the cluster that you are deleting.

This example confirms that the cluster `adccluster` is stopped and contains no instances and deletes the cluster `adccluster`.

```
asadmin> list-clusters adccluster
adccluster not running
Command list-clusters executed successfully.
asadmin> list-instances adccluster
Nothing to list.
Command list-instances executed successfully.
asadmin> delete-cluster adccluster
Command delete-cluster executed successfully.
```

See Also

- to-stop-a-cluster
- to-delete-an-instance-centrally
- to-delete-an-instance-locally
- "`delete-cluster`"
- "`list-clusters`"
- "`list-instances`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help delete-cluster`
- `asadmin help list-clusters`
- `asadmin help list-instances`

Administering Deployment Groups

Deployment Groups were introduced in Azul Payara 5 as an effective replacement for "clusters". Deployment groups are a complement to the "Data Grid" whereas the grid is responsible to managing distributed application data across multiple instances, deployment groups manage how application deployment is targeted in general.

A Deployment Group is a loose grouping of Payara Server instances that can be used as a target for **deployments**. Deployment Groups consist of one or more Payara Server types-of-payara-server-instances and each instance can belong to one or more deployment groups.

Deployment Groups can then be used to manage the lifecycle of the standalone instances in the group. All instances can be started, re-started and stopped via their corresponding deployment group.

Deployment Group Benefits

With the introduction of Deployment Groups and the Domain Data Grid, Payara Server instances are the primary resource for management: First, you create the managed servers you need running on various nodes in your environment. You can then add these instances to various deployment groups depending on your business requirements.

For example, imagine you are in a cloud environment, and you have a couple of deployment groups:

- `core-web` running your core e-commerce system
- `reporting` running applications for management reporting.

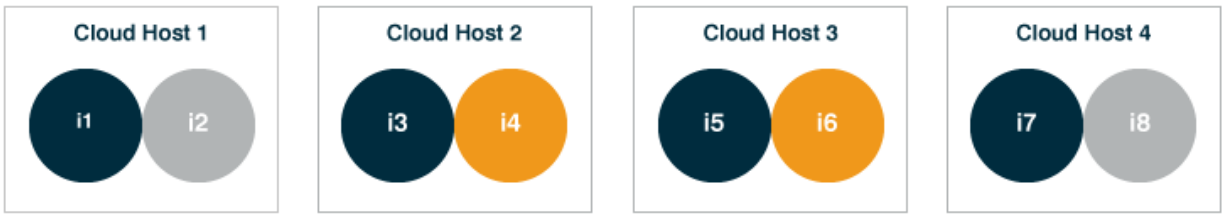
With deployment groups in response to peak load it is very simple to re-purpose a reporting instance to the core-web deployment group with a single asadmin command

like this:

```
asadmin add-instance-to-deployment-group --instance reporting-1 --deploymentgroup
core-web
```

For another example in a cloud environment (shown below), you could have each IAAS virtual server configured to have a couple of generic Payara Server instances running on them (colored grey). Then, when you come to deploy applications, you could select a number of these instances to be in deployment group 1 (colored blue), a number in deployment group 2 (colored orange) and so on.

With the flexibility to change the distribution as time goes on. As data sharing of sessions is spread throughout the whole Payara Server domain, moving instances between deployment groups does not impact accessibility of web session data.



Managing Deployment Groups

Deployment groups can be created, modified, listed and deleted from both the Admin Console and the Asadmin CLI.

Creating a Deployment Group

Once a new deployment group is created, it will be created in an empty state, as standalone instances will have to be added later.

Using the Administration Console

To create a new deployment group using the Admin Console, follow these steps:

1. Navigate to the *Deployment Groups* node in the navigation tree.
2. Click on the *New* button and give the deployment group a unique name.

3. Alternatively, check the *Auto Name* box to give the deployment group a random name.
4. From this screen it is also possible to add a number of new server instances if desired:

IMPORTANT	Instances created in this screen will be created alongside a corresponding "configuration" object that is a copy of the `default-config` object. If you want these instances to reference an existing configuration object, it is better to create them from the *Instances* screen and add them to the deployment group later.

Using Asadmin Commands

To create a new deployment group run the "`create-deployment-group`" command:

```
asadmin create-deployment-group test
```

Deleting a Deployment Group

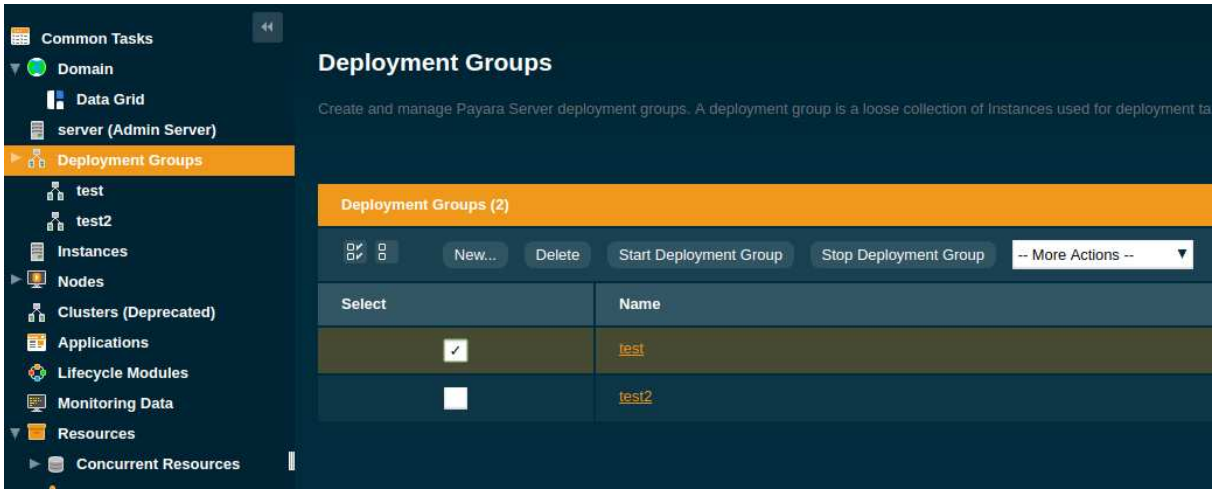
A deployment group can be deleted using both the Administration console and from the command line.

IMPORTANT	Instances that belong to a deployment group won't be deleted when the group is deleted.

Using the Administration Console

To delete a deployment group using the Admin Console, follow these steps:

1. Navigate to the *Deployment Groups* section in the side panel.
2. Select the group from corresponding table.
3. Click the *Delete* button.



Using Asadmin Commands

To delete a deployment group use the "`delete-deployment-group`" command:

```
asadmin> delete-deployment-group test
```

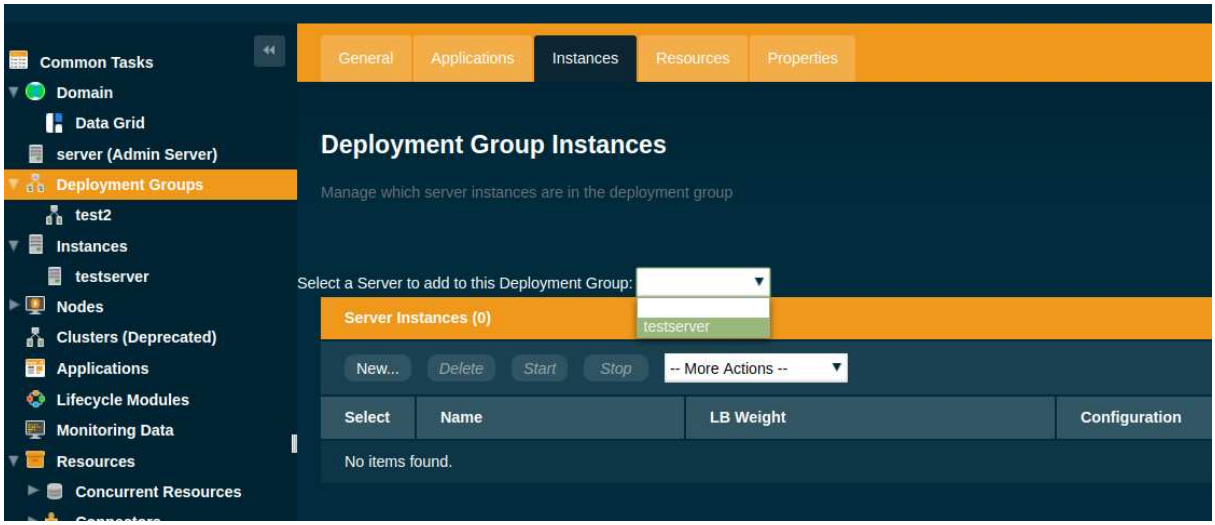
Managing Deployment Group Membership

Deployment group membership can also be managed by either using the Admin console or the Asadmin CLI.

Adding an Instance

Existing standalone instances be assigned to an existing deployment group via the Admin console by following these steps:

1. Navigate to the *Deployment Groups* node in the navigation tree.
2. Navigate to the deployment group you want to modify.
3. Select the instance to add from the dropdown in the *Instances* tab.



The same result can be achieved from the command line by using the "`add-instance-to-deployment-group`" command:

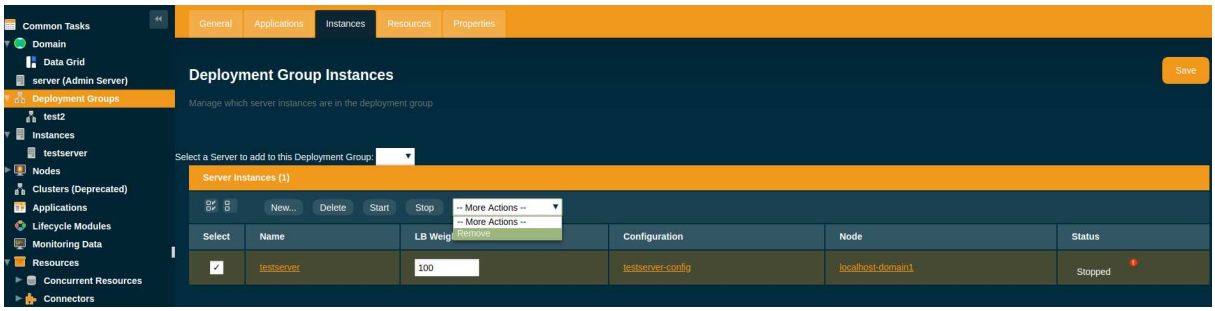
```
asadmin add-instance-to-deployment-group --instance testserver --deploymentgroup
test2
```

TIP | Remember that an instance can belong to multiple deployment groups.

Removing an Instance

If you want to remove an instance from a deployment group by using the Admin console follow these steps:

1. Navigate to the *Deployment Groups* node in the navigation tree.
2. Navigate to the deployment group you want to modify.
3. Select the instance and choose the *Remove* from the *More actions* drop down.



The same can be achieved using the Asadmin CLI by using the "`remove-instance-`

`from-deployment-group`" command:

```
asadmin remove-instance-from-deployment-group --instance testserver
--deploymentgroup test2
```

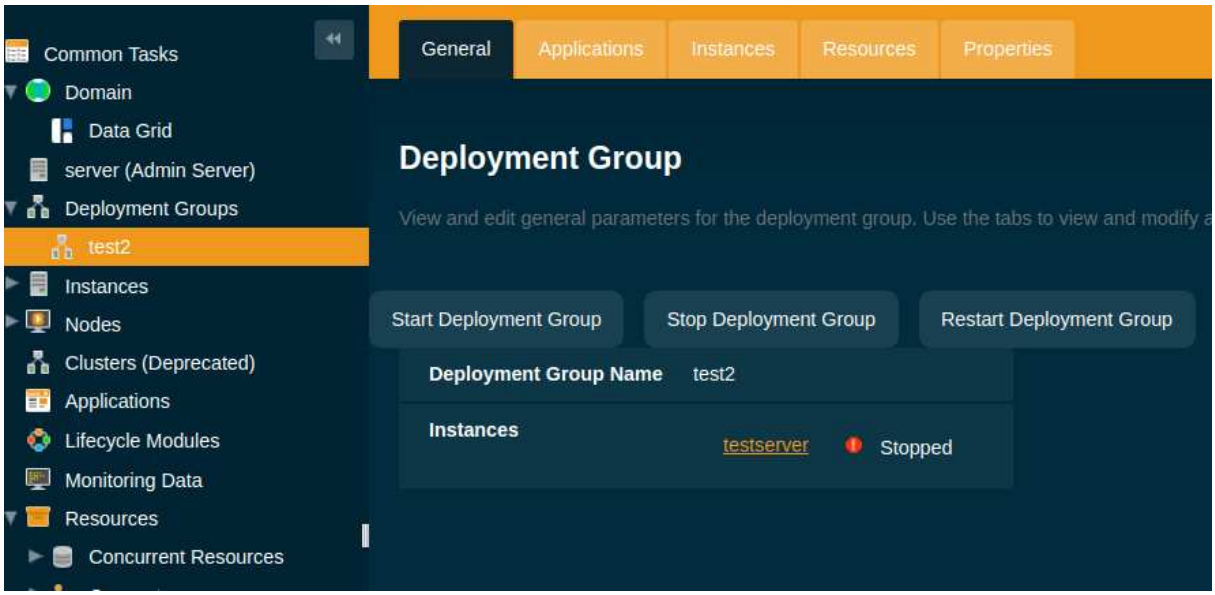
Server Lifecycle Management

Deployment Groups also provide lifecycle management capabilities. You can start a deployment group and all instances currently in the group will be started. Similarly, you can stop a deployment group. In addition, Deployment Groups support a "**rolling restart**" which is Data Grid aware.

In a rolling restart, all instances in the group are stopped and then started again in turn, ensuring that the Data Grid has enough time to adapt and ensure no loss of data during the restart.

To do this from the Admin console, follow these steps:

1. Navigate to the *Deployment Groups* node in the navigation tree.
2. Navigate to the deployment group you want to manage.
3. Choose either the *Start Deployment Group*, *Stop Deployment Group* or *Restart Deployment Group* action.



TIP	These commands are Data Grid safe and operated in a way as to minimize the danger of data disruption.

The same can be achieved from the command line by using the following Asadmin commands:

- "`start-deployment-group`"
- "`stop-deployment-group`"
- "`restart-deployment-group`"

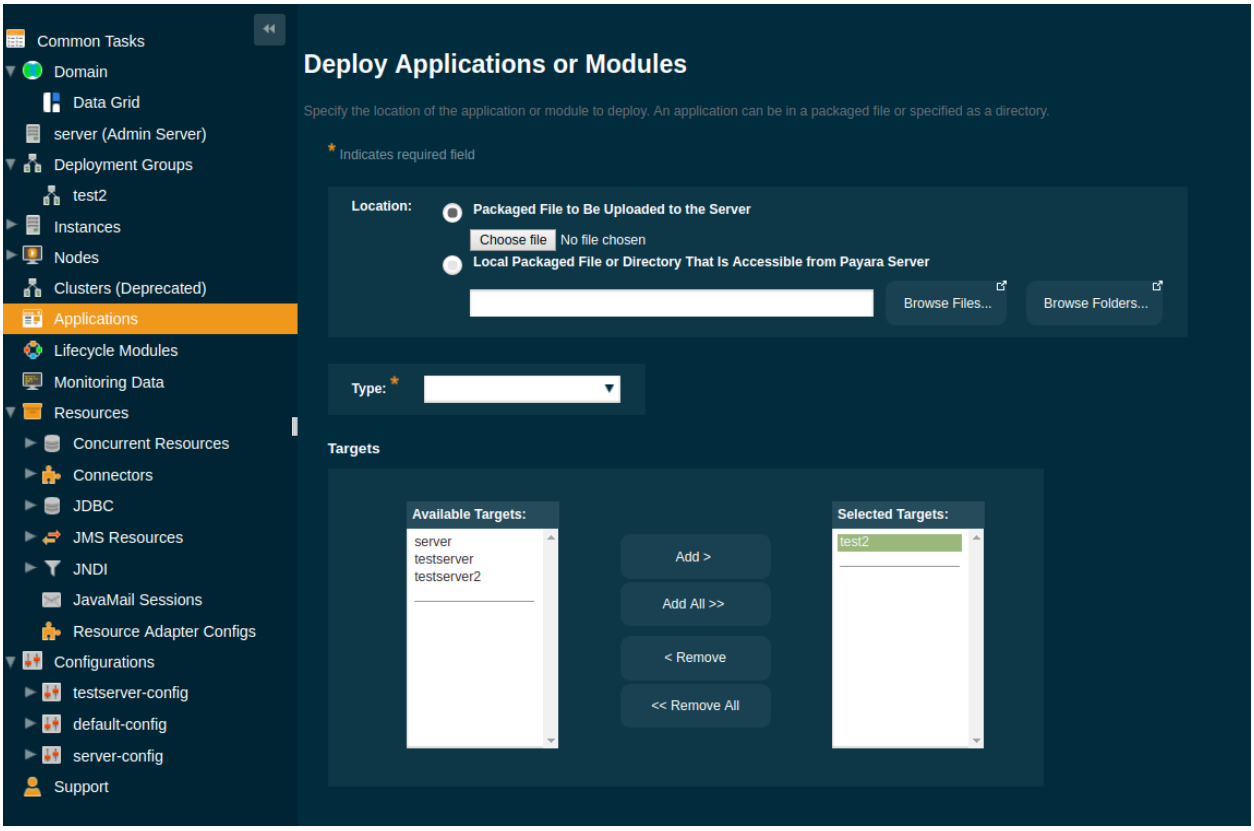
Like this:

```
asadmin start-deployment-group test2
asadmin restart-deployment-group test2
asadmin stop-Deployment Group.adoc test2
```

Deploying Applications to a Deployment Group

To deploy an application to a deployment group when using the Admin console you select the deployment group name in the list of targets in the usual way.

This can be done both on the *Applications* tab of the corresponding deployment group or on the *Applications* section of the administration console:



To deploy an application from the command line by using the "`deploy`" command, just specify the deployment group as a **target** like this:

```
asadmin deploy --target test2 test.war
```

NOTE	The same targeting also works for server resources (connection pools, mail resources, JMS resources, etc.)

See Also

- "Domain Data Grid"
- "`create-deployment-group`"
- "`delete-deployment-group`"
- "`start-deployment-group`"
- "`stop-deployment-group`"
- "`restart-deployment-group`"

- "`remove-instance-from-deployment-group`"
- "`add-instance-to-deployment-group`"

Administering the Domain Data Grid

The Domain Data Grid uses Hazelcast as its clustering technology to provide an in-memory data structure that is distributed amongst all the Payara Server instances within a Payara Domain.

The Data Grid is highly available and highly scalable and enables in-memory data storage and sharing between all Payara Server instances in a domain. The Domain Data Grid is used as the backing store for web session data, Stateful Session Bean state, "JCache" caches, Clustered Singletons, and Single Sign On state as well as internal Payara domain data like performance metrics and traces. In addition, Persistent EJB Timers can also be configured to be stored in the grid. The Domain Data Grid also enables light-weight messaging between server instances and is the foundation of the CDI Clustered Event Bus.

The Domain Data Grid greatly simplifies traditional clustering as by default it is always on, always available and usually requires no special configuration.

The Domain Data Grid has been designed to work out of the box in a variety of network topologies and usually requires zero configuration in LAN, Amazon AWS, Google Cloud Platform and Microsoft Azure.

The following topics are addressed here:

- "Domain Data Grid Benefits"
- "Domain Data Grid Discovery Modes"
- "Enabling and Disabling the Data Grid"
- "Domain Data Grid's Hazelcast settings"
- "Viewing Data Grid Members"
- "Data Grid Encryption"

Domain Data Grid Benefits

High Availability

The Domain Data Grid achieves high availability for data storage by maintaining a replica on another Payara Server Instance of any data placed into the Domain Data Grid. This replica will be placed, as a preference, on an instance that is not running on the same host as the primary data item. If the instance holding the primary copy of a piece of data fails then the backup copy will become the primary and another instance will be selected to hold a back up of the data.

Scalability

The Domain Data Grid provides high scalability for data placed into the Data Grid. As a primary and a secondary copy of any data item placed into the grid the more instances that are added to the Payara Domain the more JVM heap is available for storage of data in the grid. The Domain Data Grid can scale to 10s and 100s of instances allowing large data sets to be stored in memory. Read and Write performance for data in the grid is not dependent on the number of Payara Server instances in the domain providing scalability to grids containing large numbers of Payara Server instances.

Elasticity

The Domain Data Grid does require configuring with a specific number of Payara Server instances. Payara Server and Payara Micro instances can be added and removed and the Data Grid will shrink and grow as required. This meets the needs of cloud and container based architectures which can scale up or scale down the number of Payara instances handling requests.

Payara Micro Interoperability

The Domain Data Grid is designed to be interoperable with Payara Micro and Payara Micro instances can join the Data Grid and store data. Payara Micro has a specific cluster mode which can be enabled to join a Domain Data Grid.

Data Sharing and Encryption

The Domain Data Grid allows multiple Payara Server or Payara Micro instances to join and form a robust cluster of interchangeable nodes that can share and transfer data

between applications. Enable encryption and data stored in-memory via the Domain Data Grid will have end-to-end encryption of the data across all instances in the domain. End-to-end encryption is encryption of data both at rest (in storage) and in transit (during communication).

Light-Weight Domain Messaging

Typically, you normally need a 'system' between the sender and receiver to transport data from one domain to another and possibly hold that data until receiver is able to process it. The Domain Data Grid is a lightweight messaging transport system, so you don't need any additional resources and installation of components to send messages within the domain.

Little to No Configuration Needed

The Domain Data Grid simplifies clustering and works out-of-the-box – it's enabled by default and little to no configuration is needed for Amazon AWS, Google Cloud, Microsoft Azure, Kubernetes, and LAN environments.

Domain Data Grid Discovery Modes

The Domain Data Grid has the concept of auto-discovery. When a Payara Server or Payara Micro instance starts it must discover other Data Grid nodes so that it can join the grid. Once an instance has connected to a Payara Instance that is already a member of the grid then it can become a member of the grid. The below are the discovery mechanisms or modes that are included by default in Payara Server.

Domain Discovery Mode

The Domain discovery mode is the default discovery mode in Payara Server. In this mode when a Payara Server instance starts it uses information encoded into the `domain.xml` describing the node IP Address and ports of the other servers in the domain and the DAS to connect to an existing member of the Data Grid. For a Payara Server instance the first server it will try to connect to is the DAS if that fails it will then try each of the other instances in turn. When the DAS starts it tries to connect to each instance in turn.

NOTE

As the `domain.xml` is used to discover other members of the Data Grid no further user configuration is usually necessary.

TCP-IP Discovery Mode

The TCP-IP discovery mode is an alternative discovery mode. In this mode the user specifies a comma separated list of IP addresses and ports that a Payara Server instance should try to find members of the data grid at upon boot.

TIP

This discovery mode can be used to build a topology based on single instances of the DAS joining together to create the Data Grid. This is common in container based topologies with no central domain administration server.

Multicast Discovery Mode

The Multicast discovery mode is an alternative discovery mode. In this mode the user specifies the multicast group and multicast port used to find other members of the Data Grid. When a Payara Server instance configured with multicast discovery starts it broadcasts a message on the multicast group and port to discover other members of the data grid.

TIP

This discovery mode can be used to also build a topology based on single instances of the DAS joining together to create the data grid. In this mode multicast must be supported on the network.

NOTE

This mode is the default mode for Payara Micro

CAUTION

Also change the Data Grid group name and password to prevent other instances on the same LAN inadvertently joining your data grid.

DNS Discovery Mode

The DNS discovery mode is an alternative discovery mode, closely related to the TCP-IP discovery mode. In this mode the user specifies a comma separated list of DNS names

and ports that a Payara Server should try to find members of the data grid at upon boot.

Kubernetes Discovery Mode

The Kubernetes discovery mode is an alternative discovery mode intended for use when running in a Kubernetes environment. In this mode, the user specifies the Kubernetes service name and namespace that Payara Server should look to find other data grid members in - Payara Server will then contact the Kubernetes master upon boot to obtain the list of IP addresses of any other running instances under the specified namespace and service name.

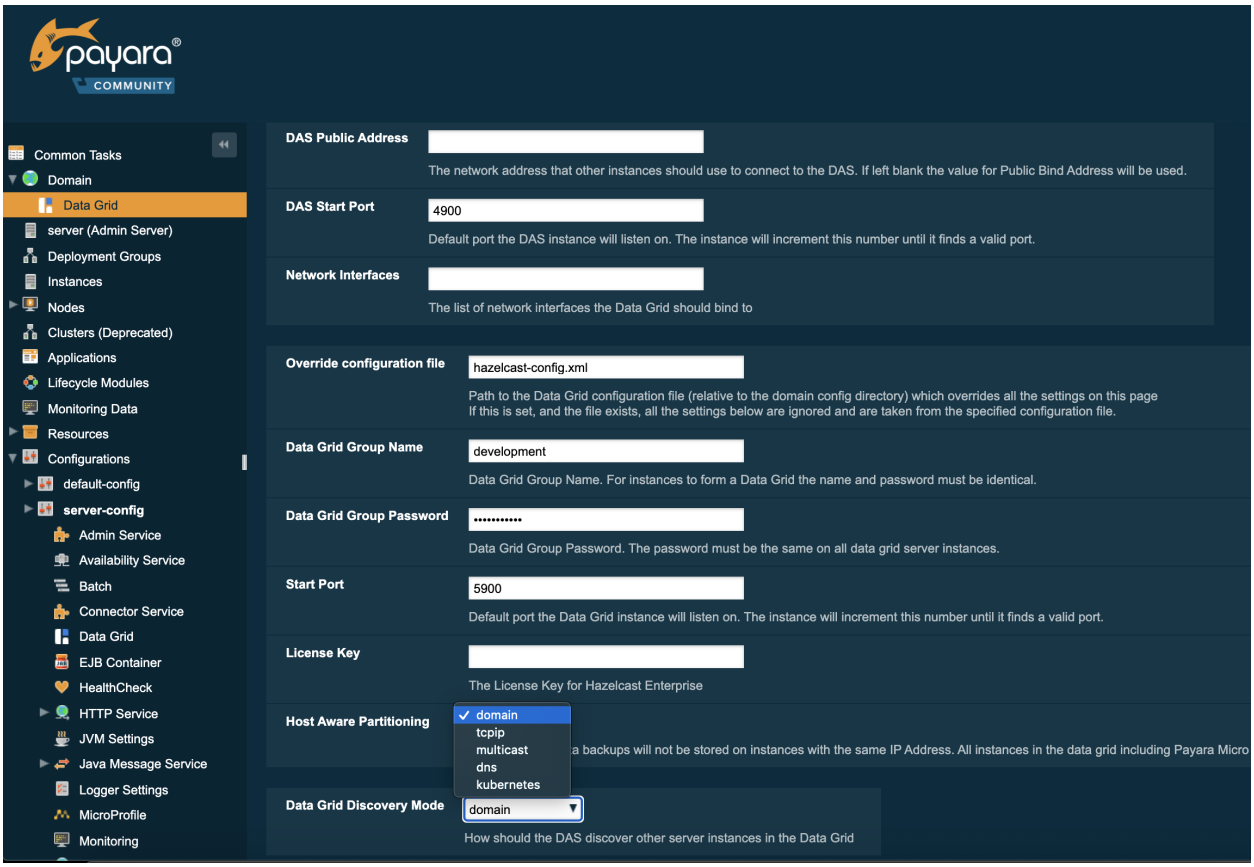
NOTE

This clustering is done using the Hazelcast Kubernetes plugin, and so you will need to grant Hazelcast permission to query the Kubernetes master node. As per the plugin documentation, you will need to apply the permissions outlined below using `kubectl apply`.
The original documentation on configuring these permissions can be found here, as well as extra documentation on how to restrict these permissions further here.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: default-cluster
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: view
subjects:
- kind: ServiceAccount
  name: default
  namespace: default
```

Configuration Discovery Mode in Administration Console

The domain discovery mode can be configured in the administration console under the **Data Grid** menu option. The discovery mode that is set gets to be applied in a domain-wide manner.



Configuration Discovery Mode via Asadmin CLI

The discovery mode can be set via the asadmin CLI, depending on the specific mode to be selected, by running the "`set-hazelcast-configuration`" commands.

Set TCP-IP Mode

When setting TCP-IP mode the list of IP addresses and ports of all Data Grid members must be specified:

```
asadmin> set-hazelcast-configuration --clustermode tcpip --tcpipmembers
192.168.0.104:4900,192.168.0.105:5900
```

Set Domain Mode

```
asadmin> set-hazelcast-configuration --clustermode domain
```

Set Multicast Mode

When setting multicast mode the multicast group and multicast port must be also specified.

NOTE

If these settings are not specified they will default to `224.2.2.3` and `54327`

```
asadmin> set-hazelcast-configuration --clustermode multicast --multicastgroup
224.2.2.4 --multicastport 55000
```

Set DNS Mode

When setting DNS mode the list of DNS names and ports of all Data Grid members must be specified.

```
asadmin> set-hazelcast-configuration --clustermode dns --dnsmembers
localhost:4900,www.example.com:5900
```

Set Kubernetes Mode

When setting Kubernetes mode the service name and namespace must be specified using the `--kubernetesservicename` and `--kubernetesnamespace` options respectively.

NOTE

If a namespace is not specified, Payara Server will use the "default" namespace.

```
asadmin> set-hazelcast-configuration --clustermode kubernetes
--kubernetesServiceName payara --kubernetesNamespace default
```

Enabling and Disabling the Data Grid

The data grid is enabled by default in Payara Server. It can be disabled and re-enabled either through the Admin Console, or by using a command line asadmin command.

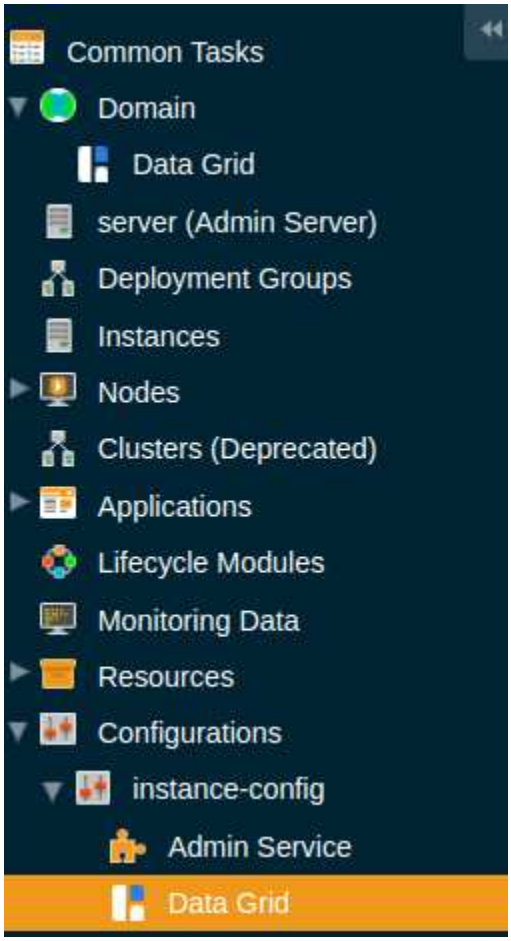
WARNING

If you are using macOS and have screen sharing enabled, Payara Server will fail to start due to screen sharing using port `5900`, the same as the default Hazelcast port.
You can disable macOS screen sharing in Preferences Sharing, Disable "Screen Sharing"`, or "change the Hazelcast start port".

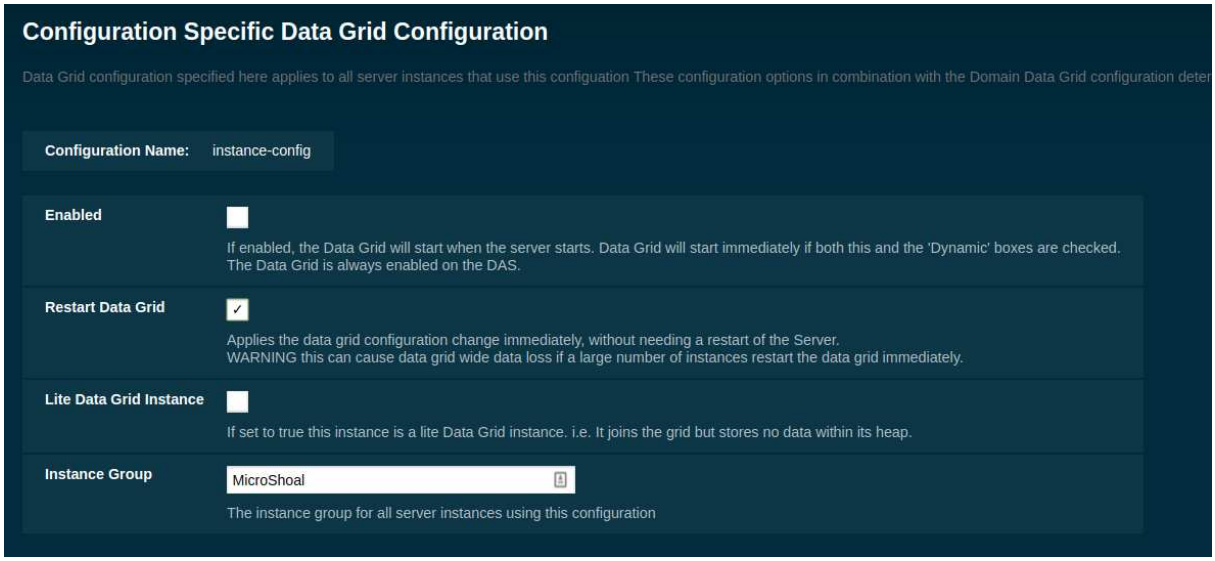
Disabling the Data Grid via the Admin Console

From the Admin Console homepage:

1. Select Data Grid under the instance's configuration from the page tree:



2. Un-check the **Enabled** option (and optionally check the **Dynamic** option to apply the configuration without the need for a restart) and save.



CAUTION

Disabling the Data Grid in a configuration object will disable the Data Grid for all instances using that same configuration. If this change is applied dynamically and multiple instances are restarted, there is big risk in incurring on data loss.
If this is a problem, ensure not to select the dynamic option and restart each affected instance manually in turn.

Disabling Datagrid Using Asadmin CLI

The "`set-hazelcast-configuration`" asadmin command can be used to enable or disable Hazelcast and therefore the Data Grid.

This command is also used to configure other aspects of Hazelcast, which is covered in this same section. The command requires the Admin Server to be running, and will expect it to be listening on port **4848** unless specified differently with the asadmin utility `--port` option.

```
asadmin> set-hazelcast-configuration --enabled=false
```

If no target is specified, the command will disable Hazelcast on the domain configuration `(server-config)` which equates to the DAS. To disable Hazelcast on another instance, configuration or cluster, use the `--target` option like so:

```
asadmin> set-hazelcast-configuration --enabled=false --target=${Target}
```

The `dynamic` option of the asadmin command defaults to true, so to enable Hazelcast and require a restart of the target instance/cluster, use `--dynamic=false`:

CAUTION	Same as above, dynamic restarting of multiple instances can incur in data loss, so keep this in mind.

```
asadmin> set-hazelcast-configuration --enabled=false --dynamic=false
```

Domain Data Grid's Hazelcast Settings

Payara Server supports configuring Hazelcast through the Admin Console and asadmin commands. The Hazelcast configuration file can also be directly edited, but will not be covered in depth in this section.

Configuration of Hazelcast is divided into four sections. The first section applies to the whole domain, the second section applies to specific configurations and therefore to Payara Server instances that use that configuration, the third section covers how certain settings can be specified when a domain is created and the final section covers how to configure config specific Data Grid Start Port during or after the creation of an instance.

Using a Hazelcast configuration file will cause the settings set via the Admin Console and *asadmin* commands to be ignored. Any parameters not specified in the configuration file reverting to the Hazelcast defaults, even if they are specified in the Admin Console or *domain.xml*

IMPORTANT	The Hazelcast defaults settings are not necessarily the same as the Payara Server's Hazelcast default settings).

NOTE	Azul Payara uses its own serializer instead of the default Hazelcast one. Setting a global serializer will not override it. This is by design, implemented to avoid a bug that causes `ClassNotFoundException` issues.

The following topics are addressed here:

- "Setting a custom Hazelcast Configuration File"
- "Configuring Server Config Specific Hazelcast Settings with the Admin Console"
- "Configuring Hazelcast on Domain Creation"
- "Configuring Config Specific Data Grid Start Port for an Instance"
- "Configuring the Hazelcast CP Subsystem"

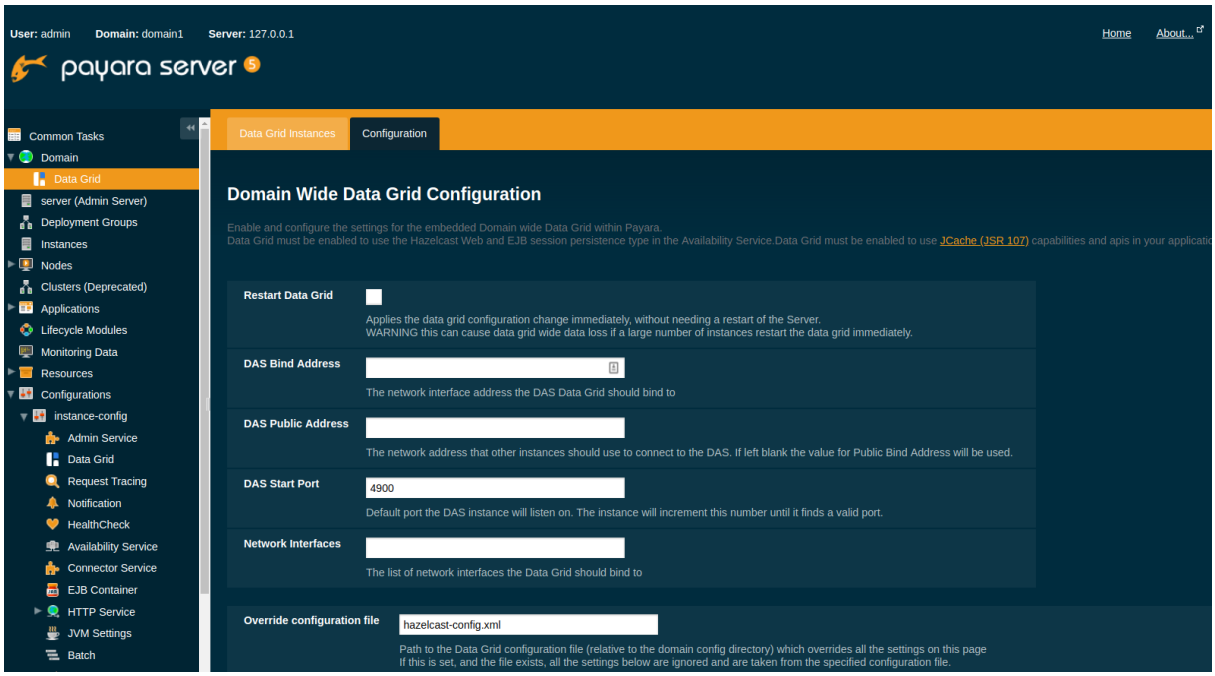
Setting a Custom Hazelcast Configuration File

A custom Hazelcast configuration file can be either set using Admin Console or asadmin CLI.

Setting a Hazelcast Configuration File with the Admin Console

To set the Hazelcast configuration from a Hazelcast Configuration file:

1. Select the domain Data Grid configuration from the page tree and click the **Configuration** tab to view the *"Hazelcast Configuration"* page:
2. On the *Hazelcast Configuration* page, add the path to your Hazelcast configuration file and click "Save". This path is relative to your domain configuration directory:



Setting a Hazelcast Configuration File Using Asadmin

The "`set-hazelcast-configuration`" command can be used to set the configuration file:

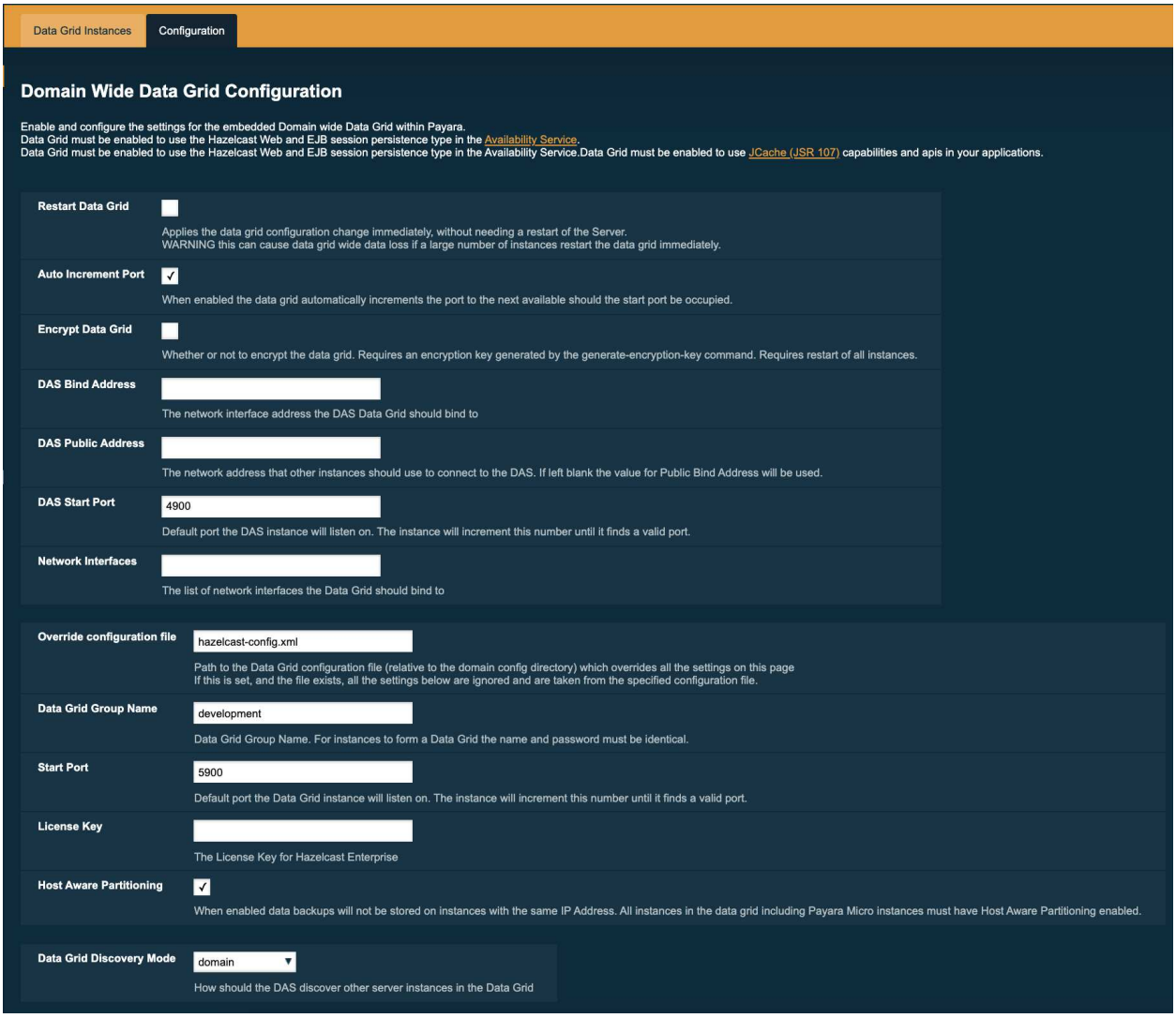
```
asadmin> set-hazelcast-configuration --hazelcastconfigurationfile /path/to/file
```

WARNING	Payara Server currently supports a custom configuration file in either XML or YAML format.

As with the admin console, this path is relative to your domain configuration directory.

Configuring Domain Wide Hazelcast Settings with the Admin Console

Some Hazelcast configuration settings apply to the whole domain. Navigate to the *Data Grid* page for the domain wide settings:



The following configuration options are available here:

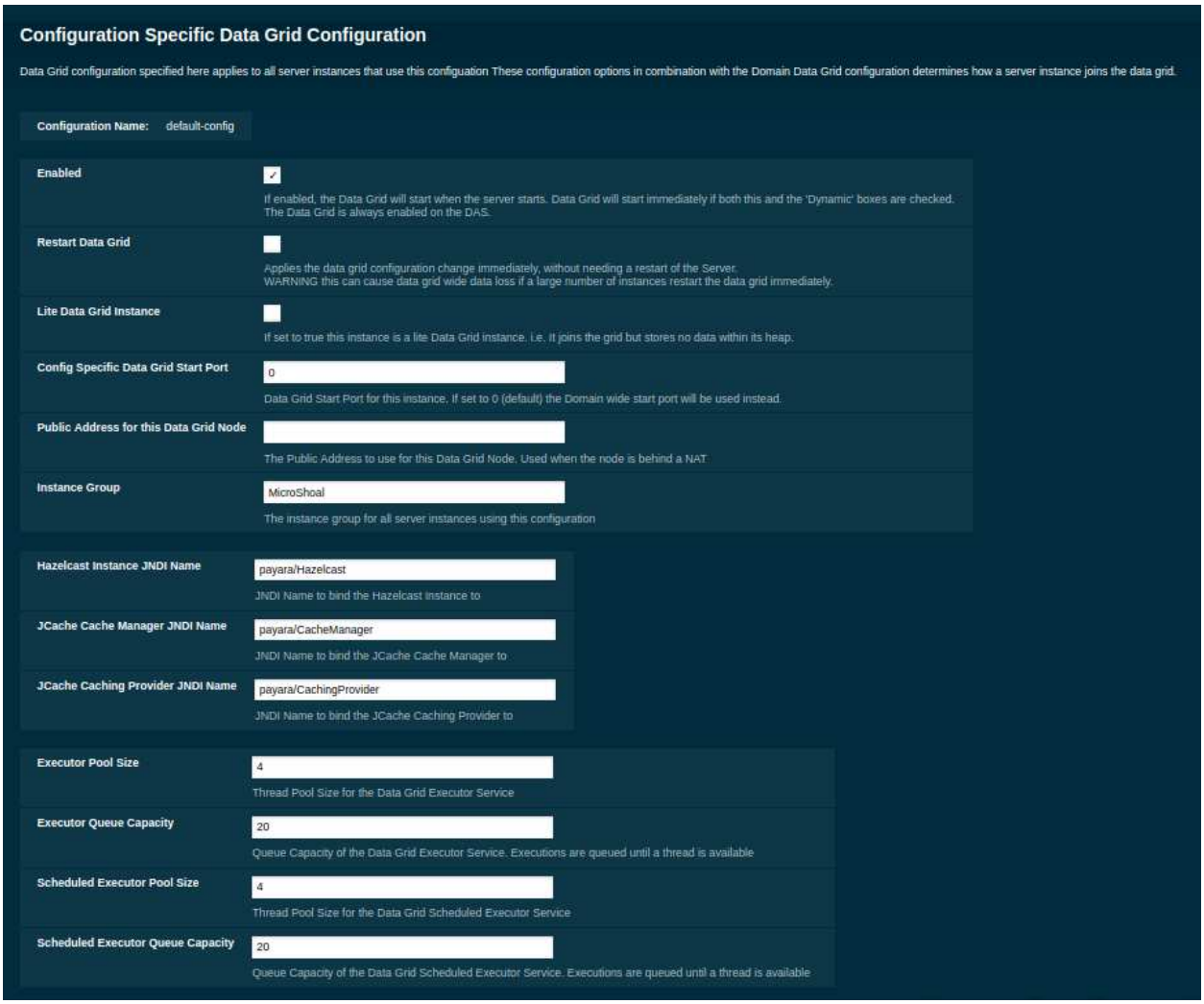
Property	Description
Restart Data Grid	Determines if the Hazelcast member embedded in Payara will be restarted to apply any changes made
DAS Bind Address	Explicitly specifies the IP Address the DAS should bind the Hazelcast instance to. By default, Hazelcast will try to choose the correct bind address from the IP addresses available on the host. If you find the DAS is binding to an incorrect IP Address the correct address can be specified here.

Property	Description
DAS Public Address	This is the public IP Address that the DAS should use if the DAS is behind a NAT firewall and other Payara Server Instances need to connect via the public NAT address rather than the internal IP Address
Auto Increment Port	By default, the cluster uses the next unoccupied port that is available starting with the start port. When auto-increment is turned off an occupied start port results in a startup failure instead.
DAS Start Port	The port the DAS uses to run Hazelcast on. If this port is busy, the port specified will be incremented until a valid port is found.
Network Interfaces	Similar to DAS Bind address by default Hazelcast will choose the correct IP Address to bind to. If you find Hazelcast is binding to the incorrect address a comma separated list of IP Addresses can be specified here.
Override Configuration File	Specifies the Hazelcast configuration file to use. The path specified is relative to the domain *config* directory. If you are using a custom configuration for a cluster or standalone instance (e.g. *cluster-config*), then the Hazelcast configuration file should be placed in the directory with the same name (e.g. `<domain-root>/config/cluster-config`). This will ensure it is replicated to the node during startup. Using this option to point to a valid Hazelcast configuration file will cause all other options set to be ignored. Any options unset within the Hazelcast configuration file will be set to the Hazelcast defaults.
Data Grid Group Name	Specifies the Hazelcast group name, to be optionally used to help divide clusters into logical, segregated groups (i.e. *dev-group*, *prod-group*).
Start Port	The port the other Payara Server instances use to run Hazelcast on. If this port is busy, the port specified will be incremented until a valid port is found.

Property	Description
License Key	Enables Hazelcast Enterprise features.
Host Aware Partitioning	Whether to enable host-aware partitioning for the cluster. Host aware partitioning **must be** enabled on all members of the cluster (including Payara Micro instances) for this feature to work correctly. This is default in Payara Server 5.181 onwards for both Payara Server domains and Payara Micro instances.
Data Grid Discovery Mode	See domain-discovery-mode

Configuring Server Config Specific Hazelcast Settings with the Admin Console

Navigate to the *Data Grid* page for the specific configuration.



The following configuration options are available here:

Property	Description
Enabled	See enabling-and-disabling-the-data-grid for detailed information
Restart Data Grid	Determines if the Hazelcast member embedded in Payara will be restarted to apply any changes made
Lite Data Grid Instance	If set to true, cluster members with this configuration will be Hazelcast "lite" nodes, meaning they will not store any cache data locally and rely on other cluster members to store data.
Config Specific Data Grid Start Port	Sets Data Grid Start Port to the instance. If set to 0 (default) the Domain wide start port will be used instead.
Public Address for this Data Grid Node	The Public Address to use for this Data Grid Node. Used when the node is behind a NAT
Instance Group	Instance group name for all Server Instances using the configuration. Future functionality may use this name.
Hazelcast Instance JNDI Name	The JNDI name to bind the Hazelcast instance to.
JCache Manager JNDI Name	The JNDI name to bind the JCache Cache Manager to.
JCache Caching Provider JNDI Name	The JNDI name to bind the JCache Caching Provider to.
Executor Pool Size	The thread pool's size for the Hazelcast Executor service
Executor Queue Capacity	Queue Capacity of the Data Grid Executor Service. Executions are queued until a thread is available
Scheduled Executor Pool Size	The thread pool's size for the Hazelcast Scheduled Executor service
Scheduled Executor Queue Capacity	Queue Capacity of the Data Grid Scheduled Executor Service. Executions are queued until a thread is available

Enter your required values, and click *Save*. Restarting the domain or instance/cluster is not necessary for any changes made to take effect, provided that *"Dynamic"* remains set to `true`

Configuring Hazelcast on Domain Creation

The following options can be used with the "`create-domain`" command to specify

certain Hazelcast settings when a domain is created. An example of create-domain command with Hazelcast specific options:

```
asadmin> create-domain --hazelcastdasport 7900 --hazelcaststartport 8900
--hazelcastautoincrement true testDomain
```

Configuring Config Specific Data Grid Start Port for an Instance

It is possible to configure the port used by Hazelcast to bind the corresponding instance to the Data Grid by letting users set its value during or after the creation of an instance.

Configuring Config Specific Data Grid Start Port on Instance Creation

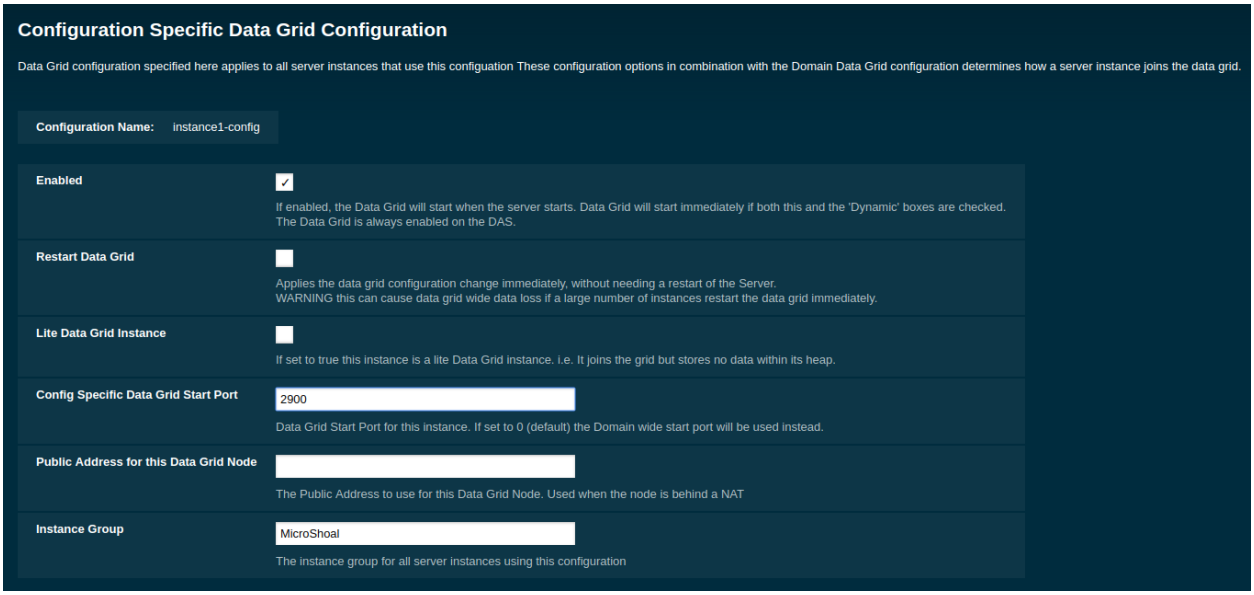
The `--dataGridStartPort` option can be used with the "`create-instance`" or the "`create-local-instance`" commands to specify config Specific Data Grid Start Port when an instance is created.

Configuring Data Grid Start Port after Instance Creation

You can configure the start port used by an instance after its creation by either using the Admin Console or the asadmin CLI.

Setting Data Grid Start Port with the Admin Console

1. Select the Configuration from the page tree, click on the relevant instance and finally click on the Data Grid.
2. On the Configuration Specific Data Grid Configuration, set the **Data Grid Start Port** and click **Save**.



Setting Data Grid Start Port Using Asadmin CLI

The "`set-hazelcast-configuration`" command can be used to set the Data Grid start port:

```
asadmin> set-hazelcast-configuration --dynamic=true
--configSpecificDataGridStartPort=2900 --target=instance1-config
```

Configuring the Hazelcast CP Subsystem

The CP Subsystem is a component of a Hazelcast cluster that builds a strongly consistent layer for a set of distributed data structures. The CP Subsystem withstands server and client failures and offer special features to Payara Server applications.

You can read more about the CP subsystem in the official Hazelcast Platform documentation.

To enable the CP subsystem, you will have to configure the `hazelcast.cp-subsystem.cp-member-count` system property to a value greater than zero (read the recommendations in the Hazelcast Platform documentation to pick a suitable member count). Hazelcast recommends that a minimum of `3` members need to be configured for the CP subsystem to work as intended.

Additionally, the `hazelcast.cp-subsystem.auto-promote` Azul Payara system property can be used to configure an auto-promote behaviour that will allow other

server instances to join the CP subsystem when detected by the DAS.

CAUTION	The CP subsystem is not actively tested as a recommended feature of Payara Server and should be used with extreme caution on highly dynamic environments.

Viewing Data Grid Members

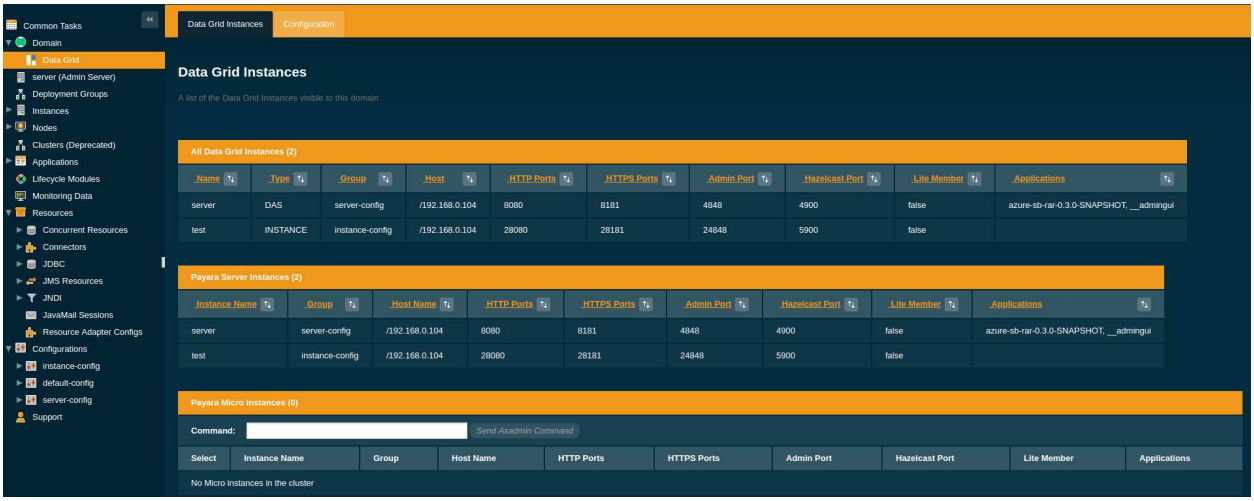
This section details how to visualize the members of the Domain Data Grid in Payara Server.

Viewing Data Grid Members from the Admin Console

To view data grid members from the admin console:

Viewing from the DAS

Selected the **Data Grid** menu option from the domain menu in the sidebar and select the *Data Grid Instances* tab:



Viewing Cluster Members via Asadmin

To view the current cluster members from the command line, run either the "`list-hazelcast-members`" or the "`list-hazelcast-cluster-members`" asadmin commands:

```
asadmin> list-hazelcast-members
\{ server-MicroShoal-/192.168.1.148:4900-this \}
```

```
asadmin> list-hazelcast-cluster-members

Instance Name  Instance Group  Instance Type  Host Name  HTTP Ports  HTTPS Ports
Admin Port  Hazelcast Port  Lite Member  Deployed Applications  Last Hearbeat
server         server-config   DAS            127.0.1.1  8080        8181
4848        5900            false        __admingui             2019-03-05 09:25:42
```

NOTE

The **Last Heartbeat** attribute will display the date and time that the DAS last received a heartbeat pulse from a Payara Server or Payara Micro instance. This heartbeat should refresh every 5 seconds.

Data Grid Encryption

Encryption of the data stored within the data grid of Hazelcast has previously required a **Hazelcast Enterprise** subscription. While this remains an option for those who would like the additional features it provides (such as WAN replication), Payara Server now includes its own encryption implementation that provides this feature to you without an additional subscription.

The following topics are addressed here:

- "Encryption Mechanism"
- "Encryption Targets"
- "Setup"
- "Generating an Encryption Key"
- "Enabling Encryption in Hazelcast"
- "Web Session Availability Configuration"
- "Stateful Session Bean Configuration"
- "Request Tracing Configuration"
- "HealthCheck Configuration"

Encryption Mechanism

This feature provides encryption of the data that's being stored in-memory via the Hazelcast data grid, providing end-to-end encryption of the data across all instances in

the domain. End-to-end encryption is encryption of data both at rest (in storage) and in transit (during communication).

Payara Server uses a symmetric key to perform the encryption of the data within the data grid store. This means that the same key is used by each member of the data grid to encrypt and decrypt the data.

This key is generated and protected using the master password and a key derivation function. The key itself is a 256-bit AES key, utilising an initialisation vector and a salt for each call of encryption to help prevent any statistical pattern recognition.

Encryption Targets

The following data stores are encrypted:

- Web Session Persistence
- Stateful Session Bean Persistence
- Request Traces
- Historic Health Checks

Setup

This section details the necessary steps to set up and enable encryption of the data grid and the various data stores.

Documentation around configuration of the data sources for encryption assumes that you've already generated the encryption key and enabled encryption of the data grid.

Generating an Encryption Key

The key to be used by Payara Server to perform the actual encryption and decryption of data can be generated using the "generate-encryption-key" asadmin command:

```
asadmin> generate-encryption-key
```

This command generates a key using the master password – the same one used to access the key store used by Payara Server. Similar to the "`change-master-`

`password`" command, this command requires you to shut down the DAS first.

IMPORTANT	This command should only be run on the DAS since this is the *master* instance, and will be the instance propagating the key out to other instances in the domain.

The default value for the master password is `changeit`. You can also provide the master password via a password file like so:

```
asadmin> -W /path/to/passwordfile.txt generate-encryption-key
```

That will generate a new file with the following contents:

```
AS_ADMIN_MASTERPASSWORD=changeit
```

NOTE	If you change the master password after generating your key, you'll need to regenerate and propagate the new key to each of your instances since the original key was created using the old (and now incorrect) password. This should occur under the default `--sync normal`, but `--sync full` can be used to delete the local copy and resynchronize it.

Enabling Encryption in Hazelcast

Enabling encryption of the data grid itself is controlled by the "`set-hazelcast-configuration`" asadmin command, with the `--encryptdatagrid` parameter:

```
asadmin> set-hazelcast-configuration --encryptdatagrid true
```

This is a domain-wide setting, and thus means that hybrid setups where encryption is enabled only on specific clusters within a domain is not supported.

IMPORTANT	Once you've generated your key and enabled encryption, you must restart all instances in the domain if they were still running; changes in encryption settings do not take effect until a server is

> restarted.
> It is **highly** recommended that you restart all of your running instances as quickly as possible, since those that haven't been restarted will be unable to decrypt the encrypted data placed in the data grid store by those that have been.

Web Session Availability Configuration

Encrypting web session availability data requires no extra actions on top of configuring the Web Container Availability Persistence Type to *hazelcast* (which it is by default).

```
asadmin> set configs.config.server-config.availability-service.web-container-
availability.persistence-type=hazelcast
```

Stateful Session Bean Configuration

Encrypting Stateful Session Beans (SFSB) availability data requires no extra actions on top of configuring the EJB Container Availability HA and/or SFSB Persistence Type to `hazelcast`:

```
asadmin> set configs.config.server-config.availability-service.ejb-container-
availability.sfsb-ha-persistence-type=hazelcast
asadmin> set configs.config.server-config.availability-service.ejb-container-
availability.sfsb-persistence-type=hazelcast
```

Request Tracing Configuration

No additional actions necessary.

Healthcheck Configuration

No additional actions necessary.

The Hazelcast Phonehome Service

In a similar manner to the server's phone-home service, Hazelcast Community Edition also comes with an internal service that makes all instances that belong to the DataGrid send phone data regularly.

As with the Platform's phone home service, if you want to disable Hazelcast Phone

Home service, it can be done via either the `HZ_PHONE_HOME_ENABLED` environment variable (set it to `false`) or by using the `hazelcast.phone.home.enabled` system property:

```
asadmin create-jvm-options --target server-config "-
Dhazelcast.phone.home.enabled=false"
```

See Also

- "Deployment Groups",
- "`create-domain`",
- "`create-instance`",
- "`create-local-instance`",
- "`set-hazelcast-configuration`",
- "`list-hazelcast-cluster-members`",
- "`list-hazelcast-members`",
- "`restart-hazelcast`",
- "`list-cache-keys`",
- "`list-caches`",
- "`clear-cache`",
- "`set-hazelcast-configuration`",
- "`get-hazelcast-configuration`",
- "`generate-encryption-key`"

Administering Payara Server Instances

A Payara Server instance is a single Virtual Machine for the Java platform (Java Virtual Machine or JVM machine) on a single node in which Payara Server is running. A node defines the host where the Payara Server instance resides.

NOTE	The JVM machine must be compatible with the server's matching

> Jakarta Platform, Enterprise Edition (Jakarta EE) version.

Payara Server instances form the basis of an application deployment. An instance is a building block in the clustering, load balancing, and session persistence features of Payara Server. Each instance belongs to a single domain and has its own directory structure, configuration, and deployed applications. Every instance contains a reference to a node that defines the host where the instance resides.

Types of Payara Server Instances

Each Payara Server instance is one of the following types of instance:

Standalone Instance: A standalone instance does not share its configuration with any other instances or clusters. A standalone instance is created if either of the following conditions is met:

+ * No configuration or cluster is specified in the command to create the instance. * A configuration that is not referenced by any other instances or clusters is specified in the command to create the instance.

+ When no configuration or cluster is specified, a copy of the `default-config` configuration is created for the instance.

+ The name of this configuration is `instance-name-config`, where instance-name represents the name of an un-clustered server instance.

Shared instance: A shared instance shares its configuration with other instances or clusters. A shared instance is created if a configuration that is referenced by other instances or clusters is specified in the command to create the instance.

Clustered instance: A clustered instance inherits its configuration from the cluster to which the instance belongs and shares its configuration with other instances in the cluster.

+ A clustered instance is created if a cluster is specified in the command to create the instance.

+ Any instance that is not part of a cluster is considered an un-clustered server instance. Therefore, standalone instances and shared instances are un-clustered server instances.

Docker instance: A Docker instance denotes an instance that is running within a Docker container and offers flexibility, portability, and scalability for deploying Payara Server applications in containerized environments.

Administering Payara Server Instances Centrally

Centralized administration requires a secure shell (SSH) to be set up. If SSH is set up, you can administer clustered instances without the need to log in to hosts where remote instances reside.

For information about setting up SSH, see enabling-centralized-administration-of-payara-server-instances.

Administering Payara Server instances centrally involves the following tasks:

To Create an Instance Centrally

Use the `create-instance` subcommand in remote mode to create a Payara Server instance centrally. Creating an instance adds the instance to the DAS configuration and creates the instance's files on the host where the instance resides.

- If the instance is a clustered instance that is managed by GMS, system properties for the instance that relate to GMS must be configured correctly.
- To avoid the need to restart the DAS and the instance, configure an instance's system properties that relate to GMS when you create the instance.
- If you change GMS-related system properties for an existing instance, the DAS and the instance must be restarted to apply the changes.

Before you begin, ensure that following prerequisites are met:

- The node where the instance is to reside exists.

- The node where the instance is to reside is either enabled for remote communication or represents the host on which the DAS is running.

For information about how to create a node that is enabled for remote communication, see the following sections:

- to-create-an-ssh-node
    - The user of the DAS can use SSH to log in to the host for the node where the instance is to reside.

If any of these prerequisites is not met, create the instance locally as explained in to-create-an-instance-locally.

If you are adding the instance to a cluster, ensure that the cluster to which you are adding the instance exists. For information about how to create a cluster, see to-create-a-cluster.

If the instance is to reference an existing named configuration, ensure that the configuration exists. For more information, see to-create-a-named-configuration.

The instance might be a clustered instance that is managed by GMS and resides on a node that represents a multi-home host.

In this situation, ensure that you have the Internet Protocol (IP) address of the network interface to which GMS binds.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the `create-instance` subcommand.

NOTE	Only the options that are required to complete this task are provided in this step. For information about all the options for configuring the instance, see the "`create-instance`" help page.

- If you are creating a standalone instance, do not specify a cluster.

If the instance is to reference an existing configuration, specify a configuration that no other cluster or instance references.

```
asadmin> create-instance --node node-name [--config configuration-name]
instance-name
```

node-name: The node on which the instance is to reside. **configuration-name**: The name of the existing named configuration that the instance will reference.

If you do not require the instance to reference an existing configuration, omit this option. A copy of the `default-config` configuration is created for the instance. The name of this configuration is instance-name`-config`, where instance-name is the name of the server instance. **instance-name**: Your choice of name for the instance that you are creating.

- If you are creating a shared instance, specify the configuration that the instance will share with other clusters or instances.

```
asadmin> create-instance --node node-name --config configuration-name instance-
name
```

node-name: The node on which the instance is to reside. **configuration-name**: The name of the existing named configuration that the instance will reference. **instance-name**: Your choice of name for the instance that you are creating.

- If you are creating a clustered instance, specify the cluster to which the instance will belong.

  If the instance is managed by GMS and resides on a node that represents a multi-home host, specify the `GMS-BIND-INTERFACE-ADDRESS-`cluster-name system property.

```
asadmin> create-instance --cluster cluster-name --node node-name [
--systemproperties GMS-BIND-INTERFACE-ADDRESS-cluster-name=bind-
address]instance-name
```

cluster-name: The name of the cluster to which you are adding the instance. **node-name**: The node on which the instance is to reside. **bind-address**: The IP address of the network interface to which GMS binds. Specify this option only if the instance is managed by GMS and resides on a node that represents a multi-home host. **instance-name**: Your choice of name for the instance that you are creating.

The following example adds the instance `pmd-i1` to the cluster `pmdclust` in the domain `domain1`. The instance resides on the node `sj01`, which represents the host `sj01.example.com`.

```
asadmin> create-instance --cluster pmdclust --node sj01 pmd-i1
Port Assignments for server instance pmd-i1:
JMX_SYSTEM_CONNECTOR_PORT=28686
JMS_PROVIDER_PORT=27676
HTTP_LISTENER_PORT=28080
ASADMIN_LISTENER_PORT=24848
IIOP_SSL_LISTENER_PORT=23820
IIOP_LISTENER_PORT=23700
HTTP_SSL_LISTENER_PORT=28181
IIOP_SSL_MUTUALAUTH_PORT=23920
The instance, pmd-i1, was created on host sj01.example.com
Command create-instance executed successfully.
```

See Also

- to-create-an-ssh-node
- to-create-an-instance-locally
- create-instance

You can also view the full syntax and options of the subcommand by typing `asadmin help create-instance` at the command line.

To List All Instances in a Domain

Use the `list-instances` subcommand in remote mode to obtain information about existing instances in a domain.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the "`list-instances`" subcommand.

```
asadmin> list-instances
```

The following example lists the name and status of all Payara Server instances in the current domain.

```
asadmin> list-instances
pmd-i2 running
yml-i2 running
pmd-i1 running
yml-i1 running
pmdsa1 not running
Command list-instances executed successfully.
```

The following example lists detailed information about all Payara Server instances in the current domain:

```
asadmin> list-instances --long=true
NAME    HOST              PORT   PID    CLUSTER     STATE
pmd-i1  sj01.example.com  24848  31310  pmdcluster   running
yml-i1  sj01.example.com  24849  25355  ymlcluster   running
pmdsa1  localhost         24848  -1     ---          not running
pmd-i2  sj02.example.com  24848  22498  pmdcluster   running
yml-i2  sj02.example.com  24849  20476  ymlcluster   running
ymlsa1  localhost         24849  -1     ---          not running
Command list-instances executed successfully.
```

See Also

"`list-instances`"

You can also view the full syntax and options of the subcommand by typing `asadmin help list-instances` at the command line.

To Delete an Instance Centrally

Use the `delete-instance` subcommand in remote mode to delete a Payara Server instance centrally.

WARNING	If you are using a Java Message Service (JMS) cluster with a master broker, do not delete the instance that is associated with the master broker. If this instance must be deleted, use the "`change-master-`

> `broker`" subcommand first to assign the master broker to a different instance.

Deleting an instance involves the following:

- Removing the instance from the configuration of the DAS
- Deleting the instance's files from file system

Ensure that the instance that you are deleting is not running. For information about how to stop an instance, see the following sections:

- to-start-an-individual-instance-centrally
- to-stop-an-individual-instance-locally

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Confirm that the instance is not running.

   ```
   asadmin> list-instances instance-name
   ```

   instance-name: The name of the instance that you are deleting.

3. Run the "`delete-instance`" subcommand.

   ```
   asadmin> delete-instance instance-name
   ```

   instance-name: The name of the instance that you are deleting.

The following example confirms that the instance `pmd-i1` is not running and deletes the instance.

```
asadmin> list-instances pmd-i1
pmd-i1   not running
Command list-instances executed successfully.
asadmin> delete-instance pmd-i1
Command _delete-instance-filesystem executed successfully.
The instance, pmd-i1, was deleted from host sj01.example.com
Command delete-instance executed successfully.
```

See Also

- to-start-an-individual-instance-centrally
- to-stop-an-individual-instance-locally
- "`change-master-broker`(1)"
- "`delete-instance`"
- "`list-instances`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help delete-instance`
- `asadmin help list-instances`

To Start a Cluster

Use the `start-cluster` subcommand in remote mode to start a cluster. Starting a cluster starts all instances in the cluster that are not already running.

Ensure that following prerequisites are met:

- Each node where an instance in the cluster resides is either enabled for remote communication or represents the host on which the DAS is running.
- The operating system user that runs the DAS can use SSH to log in to the host for any node where instances in the cluster reside.

If any of these prerequisites is not met, start the cluster by starting each instance locally as explained in to-start-an-individual-instance-locally.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the "`start-cluster`" subcommand.

```
asadmin> start-cluster cluster-name
```

cluster-name: The name of the cluster that you are starting.

The following example starts the cluster `pmdcluster`.

```
asadmin> start-cluster pmdcluster
Command start-cluster executed successfully.
```

See Also

- to-start-an-individual-instance-locally
- "`start-cluster`"

You can also view the full syntax and options of the subcommand by typing `asadmin help start-cluster` at the command line.

To Stop a Cluster

Use the `stop-cluster` subcommand in remote mode to stop a cluster.

Stopping a cluster stops all running instances in the cluster.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the "`stop-cluster`" subcommand.

   ```
   asadmin> stop-cluster cluster-name
   ```

   cluster-name: The name of the cluster that you are stopping.

The following example stops the cluster `pmdcluster`.

```
asadmin> stop-cluster pmdcluster
Command stop-cluster executed successfully.
```

See Also

"`stop-cluster`"

You can also view the full syntax and options of the subcommand by typing `asadmin`

`help stop-cluster` at the command line.

Troubleshooting

If instances in the cluster have become unresponsive and fail to stop, run the subcommand again with the `--kill` option set to `true`. When this option is `true`, the subcommand uses functionality of the operating system to kill the process for each running instance in the cluster.

To Start an Individual Instance Centrally

Use the `start-instance` subcommand in remote mode to start an individual instance centrally.

Ensure that following prerequisites are met:

- The node where the instance resides is either enabled for remote communication or represents the host on which the DAS is running.
- The operating system user that runs the DAS can use SSH to log in to the host for any node where the instance resides.

If any of these prerequisites is not met, start the instance locally as explained in to-start-an-individual-instance-locally.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the `start-instance` subcommand.

```
asadmin> start-instance instance-name
```

NOTE Only the options that are required to complete this task are provided in this step. For information about all the options for controlling the behavior of the instance, see the "`start-instance`" help page.

instance-name: The name of the instance that you are starting.

The following example starts the instance `pmd-i2`, which resides on the node `sj02`.

This node represents the host `sj02.example.com`.

The configuration of the instance on this node already matched the configuration of the instance in the DAS when the instance was started.

```
asadmin> start-instance pmd-i2
CLI801 Instance is already synchronized
Waiting for pmd-i2 to start .....
Successfully started the instance: pmd-i2
instance Location: /export/payara7/glassfish/nodes/sj02/pmd-i2
Log File: /export/payara7/glassfish/nodes/sj02/pmd-i2/logs/server.log
Admin Port: 24851
Command start-local-instance executed successfully.
The instance, pmd-i2, was started on host sj02.example.com
Command start-instance executed successfully.
```

See Also

"`start-instance`"

You can also view the full syntax and options of the subcommand by typing `asadmin help start-instance` at the command line.

To Stop an Individual Instance Centrally

Use the `stop-instance` subcommand in remote mode to stop an individual instance centrally.

When an instance is stopped, the instance stops accepting new requests and waits for all outstanding requests to be completed.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the "`stop-instance`" subcommand.

The following example stops the instance `pmd-i2`.

```
asadmin> stop-instance pmd-i2
The instance, pmd-i2, is stopped.
Command stop-instance executed successfully.
```

See Also

"`stop-instance`"

You can also view the full syntax and options of the subcommand by typing `asadmin help stop-instance` at the command line.

Troubleshooting

If the instance has become unresponsive and fails to stop, run the subcommand again with the `--kill` option set to `true`. When this option is `true`, the subcommand uses functionality of the operating system to kill the instance process.

To Restart an Individual Instance Centrally

Use the `restart-instance` subcommand in remote mode to start an individual instance centrally.

When this subcommand restarts an instance, the DAS synchronizes the instance with changes since the last synchronization as described in default-synchronization-for-files-and-directories.

If you require different synchronization behavior, stop and start the instance as explained in to-resynchronize-an-instance-and-the-das-online.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the "`restart-instance`" subcommand.

```
asadmin> restart-instance instance-name
```

instance-name: The name of the instance that you are restarting.

The following example restarts the instance `pmd-i2`.

```
asadmin> restart-instance pmd-i2
pmd-i2 was restarted.
Command restart-instance executed successfully.
```

See Also

- to-start-an-individual-instance-centrally
- to-start-an-individual-instance-centrally
- restart-instance

You can also view the full syntax and options of the subcommand by typing `asadmin help restart-instance` at the command line.

Troubleshooting

If the instance has become unresponsive and fails to stop, run the subcommand again with the `--kill` option set to `true`. When this option is `true`, the subcommand uses functionality of the operating system to kill the instance process before restarting the instance.

Administering Payara Server Instances Locally

Local administration does not require SSH to be set up. If SSH is set up, you must log in to each host where remote instances reside and administer the instances individually.

TIP	Even SSH is set up, you can obtain information about instances in a domain without logging in to each host where remote instances reside. For instructions, see to-list-all-instances-in-a-domain.

To Create an Instance Locally

Use the `create-local-instance` subcommand in remote mode to create a Payara Server instance locally. Creating an instance adds the instance to the DAS configuration and creates the instance's files on the host where the instance resides.

NOTE	If the instance is a clustered instance, system properties for the instance that relate to GMS must be configured correctly. If you change GMS-related system properties for an existing instance, the DAS and the instance must be restarted to apply the changes. For information about GMS, see group-management-service.

Before You Begin

If you plan to specify the node on which the instance is to reside, ensure that the node exists.

NOTE

If you create the instance on a host for which no nodes are defined, you can create the instance without creating a node beforehand. In this situation, Payara Server creates a `CONFIG` node for you. The name of the node is the unqualified name of the host.

For information about how to create a node, see the following sections:

- to-create-an-ssh-node
- to-create-a-config-node
- creating-a-docker-node

If the instance is to reference an existing named configuration, ensure that the configuration exists. For more information, see to-create-a-named-configuration.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Log in to the host that is represented by the node where the instance is to reside.
3. Run the `create-local-instance` subcommand.

   NOTE

   Only the options that are required to complete this task are provided in this step. For information about all the options for configuring the instance, see the "`create-local-instance`" help page.

   - If you are creating a standalone instance, do not specify a cluster name.

     If the instance is to reference an existing configuration, specify a configuration that no other cluster or instance references.

     ```
     asadmin --host das-host [--port admin-port] create-local-instance [--node
     node-name] [--config configuration-name] instance-name
     ```

     das-host: The name of the host where the DAS is running. **admin-port**: The HTTP

or HTTPS port on which the DAS listens for administration requests. If the DAS listens on the default port for administration requests, you may omit this option. **node-name**: The node on which the instance is to reside.

If you are creating the instance on a host for which fewer than two nodes are defined, you may omit this option.

If no nodes are defined for the host, Payara Server creates a CONFIG node for you. The name of the node is the unqualified name of the host.

If one node is defined for the host, the instance is created on that node. **configuration-name**: The name of the existing named configuration that the instance will reference.

If you do not require the instance to reference an existing configuration, omit this option.

A copy of the `default-config` configuration is created for the instance. The name of this configuration is `<instance-name>-config`, where instance-name is the name of the server instance. **instance-name**: Your choice of name for the instance that you are creating.

- If you are creating a shared instance, specify the configuration that the instance will share with other clusters or instances.

```
asadmin --host das-host [--port admin-port] create-local-instance [--node
node-name] --config configuration-name instance-name
```

das-host: The name of the host where the DAS is running. **admin-port**: The HTTP or HTTPS port on which the DAS listens for administration requests. If the DAS listens on the default port for administration requests, you may omit this option. **node-name**: The node on which the instance is to reside.

If you are creating the instance on a host for which fewer than two nodes are defined, you may omit this option.

If no nodes are defined for the host, Payara Server creates a `CONFIG` node for you. The name of the node is the unqualified name of the host.

If one node is defined for the host, the instance is created on that node. **configuration-name**: The name of the existing named configuration that the instance will reference. **instance-name**: Your choice of name for the instance that you are creating.

- If you are creating a clustered instance, specify the cluster to which the instance will belong.

  If the instance is managed by GMS and resides on a node that represents a multi-home host, specify the `GMS-BIND-INTERFACE-ADDRESS-`cluster-name system property.

```
$ asadmin --host das-host [--port admin-port] create-local-instance --cluster
cluster-name [--node node-name] [--systemproperties GMS-BIND-INTERFACE-
ADDRESS-cluster-name=bind-address] instance-name
```

  das-host: The name of the host where the DAS is running. **admin-port**: The HTTP or HTTPS port on which the DAS listens for administration requests.

  > **TIP** If the DAS listens on the default port for administration requests, you may omit this option. **cluster-name**: The name of the cluster to which you are adding the instance. **node-name**: The node on which the instance is to reside.

  If you are creating the instance on a host for which fewer than two nodes are defined, you may omit this option.

  If no nodes are defined for the host, Payara Server creates a `CONFIG` node for you. The name of the node is the unqualified name of the host.

  If one node is defined for the host, the instance is created on that node. **bind-address**: The IP address of the network interface to which GMS binds. Specify this

option only if the instance is managed by GMS and resides on a node that represents a multi-home host. **instance-name**: Your choice of name for the instance that you are creating.

This example adds the instance `kui-i1` to the cluster `kuicluster` locally. The `CONFIG` node `xk01` is created automatically to represent the host `xk01.example.com`, on which this example is run.

The DAS is running on the host `dashost.example.com` and listens for administration requests on the default port.

The commands to list the nodes in the domain are included in this example only to demonstrate the creation of the node `xk01`. These commands are not required to create the instance:

```
> asadmin --host dashost.example.com list-nodes --long

NODE NAME          TYPE    NODE HOST         INSTALL DIRECTORY   REFERENCED BY
localhost-domain1  CONFIG  localhost         /export/payara7
Command list-nodes executed successfully.

> asadmin --host dashost.example.com
create-local-instance --cluster kuicluster kui-i1
Rendezvoused with DAS on dashost.example.com:4848.
Port Assignments for server instance kui-i1:
JMX_SYSTEM_CONNECTOR_PORT=28687
JMS_PROVIDER_PORT=27677
HTTP_LISTENER_PORT=28081
ASADMIN_LISTENER_PORT=24849
JAVA_DEBUGGER_PORT=29009
IIOP_SSL_LISTENER_PORT=23820
IIOP_LISTENER_PORT=23700
OSGI_SHELL_TELNET_PORT=26666
HTTP_SSL_LISTENER_PORT=28182
IIOP_SSL_MUTUALAUTH_PORT=23920
Command create-local-instance executed successfully.

> asadmin --host dashost.example.com list-nodes --long
NODE NAME          TYPE    NODE HOST         INSTALL DIRECTORY   REFERENCED BY
localhost-domain1  CONFIG  localhost         /export/payara7
xk01               CONFIG  xk01.example.com  /export/payara7  kui-i1
Command list-nodes executed successfully.
```

This example adds the instance `yml-i1` to the cluster `ymlcluster` locally. The instance resides on the node `sj01`. The DAS is running on the host

`das1.example.com` and listens for administration requests on the default port.

```
> asadmin --host das1.example.com

create-local-instance --cluster ymlcluster --node sj01 yml-i1
Rendezvoused with DAS on das1.example.com:4848.
Port Assignments for server instance yml-i1:
JMX_SYSTEM_CONNECTOR_PORT=28687
JMS_PROVIDER_PORT=27677
HTTP_LISTENER_PORT=28081
ASADMIN_LISTENER_PORT=24849
JAVA_DEBUGGER_PORT=29009
IIOP_SSL_LISTENER_PORT=23820
IIOP_LISTENER_PORT=23700
OSGI_SHELL_TELNET_PORT=26666
HTTP_SSL_LISTENER_PORT=28182
IIOP_SSL_MUTUALAUTH_PORT=23920
Command create-local-instance executed successfully.
```

See Also

- to-create-an-ssh-node
- to-create-a-config-node
- "`create-local-instance`"

You can also view the full syntax and options of the subcommand by typing `asadmin help create-local-instance` at the command line.

Next Steps

After creating an instance, you can start the instance as explained in the following sections:

- to-start-an-individual-instance-centrally
- to-stop-an-individual-instance-locally

To Delete an Instance Locally

Use the `delete-local-instance` subcommand in remote mode to delete a Payara Server instance locally.

CAUTION	If you are using a Java Message Service (JMS) cluster with a master

> broker, do not delete the instance that is associated with the master broker.
> If this instance must be deleted, use the "`change-master-broker`" subcommand to assign the master broker to a different instance first.

Deleting an instance involves the following:

- Removing the instance from the configuration of the DAS
- Deleting the instance's files from file system

Before You Begin

Ensure that the instance that you are deleting is not running. For information about how to stop an instance, see the following sections:

- to-start-an-individual-instance-centrally
- to-stop-an-individual-instance-locally

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Log in to the host that is represented by the node where the instance resides.
3. Confirm that the instance is not running.

   ```
   > asadmin --host das-host [--port admin-port] list-instances instance-name
   ```

   das-host: The name of the host where the DAS is running. **admin-port**: The HTTP or HTTPS port on which the DAS listens for administration requests. If the DAS listens on the default port for administration requests, you may omit this option. **instance-name**: The name of the instance that you are deleting.

4. Run the "`delete-local-instance`" subcommand.

   ```
   > asadmin --host das-host [--port admin-port] delete-local-instance [--node
   node-name]instance-name
   ```

das-host: The name of the host where the DAS is running. **admin-port**: The HTTP or HTTPS port on which the DAS listens for administration requests. If the DAS listens on the default port for administration requests, you may omit this option. **node-name**: The node on which the instance resides. If only one node is defined for the Payara Server installation that you are running on the node's host, you may omit this option. **instance-name**: The name of the instance that you are deleting.

This example confirms that the instance `yml-i1` is not running and deletes the instance.

```
$ asadmin --host das1.example.com list-instances yml-i1
yml-i1   not running
Command list-instances executed successfully.
$ asadmin --host das1.example.com delete-local-instance --node sj01 yml-i1
Command delete-local-instance executed successfully.
```

See Also

- to-start-an-individual-instance-centrally
- to-stop-an-individual-instance-locally
- "`change-master-broker`"
- "`delete-local-instance`"
- "`list-instances`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help delete-local-instance`
- `asadmin help list-instances`

To Start an Individual Instance Locally

Use the `start-local-instance` subcommand in local mode to start an individual instance locally.

1. Log in to the host that is represented by the node where the instance resides.
2. Run the `start-local-instance` subcommand.

```
$ asadmin start-local-instance [--node node-name] instance-name
```

NOTE

Only the options that are required to complete this task are provided in this step.
For information about all the options for controlling the behavior of the instance, see the "`start-local-instance`" help page.

node-name: The node on which the instance resides. If only one node is defined for the Payara Server installation that you are running on the node's host, you may omit this option. **instance-name**: The name of the instance that you are starting.

This example starts the instance `yml-i1` locally. The instance resides on the node `sj01`.

```
$ asadmin start-local-instance --node sj01 yml-i1
Waiting for yml-i1 to start ..............
Successfully started the instance: yml-i1
instance Location: /export/payara7/glassfish/nodes/sj01/yml-i1
Log File: /export/payara7/glassfish/nodes/sj01/yml-i1/logs/server.log
Admin Port: 24849
Command start-local-instance executed successfully.
```

See Also

"`start-local-instance`"

You can also view the full syntax and options of the subcommand by typing `asadmin help start-local-instance` at the command line.

Next Steps

After starting an instance, you can deploy applications to the instance. For more information, see the "Payara Server Application Deployment section".

To Stop an Individual Instance Locally

Use the `stop-local-instance` subcommand in local mode to stop an individual instance locally.

When an instance is stopped, the instance stops accepting new requests and waits for all outstanding requests to be completed.

1. Log in to the host that is represented by the node where the instance resides.
2. Run the "`stop-local-instance`" subcommand.

```
$ asadmin stop-local-instance [--node node-name]instance-name
```

node-name: The node on which the instance resides. If only one node is defined for the Payara Server installation that you are running on the node's host, you may omit this option. **instance-name**: The name of the instance that you are stopping.

This example stops the instance `yml-i1` locally. The instance resides on the node `sj01`.

```
$ asadmin stop-local-instance --node sj01 yml-i1
Waiting for the instance to stop ....
Command stop-local-instance executed successfully.
```

See Also

"`stop-local-instance`"

You can also view the full syntax and options of the subcommand by typing `asadmin help stop-local-instance` at the command line.

Troubleshooting

If the instance has become unresponsive and fails to stop, run the subcommand again with the `--kill` option set to `true`. When this option is `true`, the subcommand uses functionality of the operating system to kill the instance process.

To Restart an Individual Instance Locally

Use the `restart-local-instance` subcommand in local mode to restart an individual instance locally.

When this subcommand restarts an instance, the DAS synchronizes the instance with changes since the last synchronization as described in default-synchronization-for-files-and-directories.

If you require different synchronization behavior, stop and start the instance as explained in to-resynchronize-an-instance-and-the-das-online.

1. Log in to the host that is represented by the node where the instance resides.
2. Run the `restart-local-instance` subcommand.

```
$ asadmin restart-local-instance [--node node-name]instance-name
```

node-name: The node on which the instance resides. If only one node is defined for the Payara Server installation that you are running on the node's host, you may omit this option. **instance-name**: The name of the instance that you are restarting.

This example restarts the instance `yml-i1` locally. The instance resides on the node `sj01`.

```
> asadmin restart-local-instance --node sj01 yml-i1
Command restart-local-instance executed successfully.
```

See Also

"`restart-local-instance`"

You can also view the full syntax and options of the subcommand by typing `asadmin help restart-local-instance` at the command line.

Troubleshooting

If the instance has become unresponsive and fails to stop, run the subcommand again

with the `--kill` option set to `true`.

When this option is `true`, the subcommand uses functionality of the operating system to kill the instance process before restarting the instance.

Administering Docker Instances

A Docker instance is the term used to refer to an instance created on a Docker node. These instances exist within their own Docker containers, with the lifecycle of Docker containers being tied to the instance it was created for.

This section will illustrate how to effectively administer these instances and correctly troubleshoot any issues that may arise on the Docker environment where they reside. Here are some considerations about the examples used below:

- `payaraDas` and `dockerHost1` are hostnames known to the local naming service (you can use an IP addresses when setting this up)
- The DAS is listening on port `4848` on `payaraDas`
- The Docker process is listening on port 2376 on `dockerHost1`
- The DAS is able to reach this location: `http://dockerHost1:2376`
- Docker containers can access this location: `https://payaraDas:4848`
- There are no Docker containers created yet, no Docker nodes and no instances registered in the DAS.
- We'll use the `/app/passwordfile.txt` file which is located on `payaraDas` to set up the DAS' admin password, with the following contents:

```
AS_ADMIN_PASSWORD=admin123
```

- The `/app/passwordfile-docker.txt` file stored on `dockerHost1` has the same contents as the previous file.

TIP	Avoid using the `payara/server-node:latest` Docker image tag, because it changes without warning. If you have the intention to use it, you

> can re-tag the image and use your own stable Docker image tag like this:

```
>> docker tag payara/server-node:latest payara/server-node:mytag
```

Managing Docker Instances

Docker instances are manageable in much the same way as any other instance. Deployment of applications, assignation to Deployment Groups, editing of their configuration, should all be done just as if the instance was hosted on a standard SSH or CONFIG node.

IMPORTANT	Please note that it is required that secure-admin be enabled for Docker instances to start (which is why a password file is mandatory when creating a Docker node).

Creating a Docker Instance

There are two ways how to create a Payara Docker instance belonging to a Domain Administration Server (DAS). You can create a Docker container node and instances on it using the DAS, but you can also create a Docker container with a temporary node and instance directly from Docker. These two use cases differ a bit in their lifecycles.

From the DAS

Creating a Docker instance from the DAS is done in exactly the same way as when creating a regular instance. First, create a Docker node, and then you can create one or more Payara Server instances that are hosted on it:

```
asadmin --passwordfile /app/passwordfile.txt create-node-docker --nodehost
dockerHost1 --dockerpasswordfile /app/passwordfile-docker.txt --dockerport 2376
DockerNode1
Command create-node-docker executed successfully.

asadmin --passwordfile /app/passwordfile.txt create-instance --node DockerNode1
DockerInstance1
Port Assignments for server instance DockerInstance1:
OSGI_SHELL_TELNET_PORT=26666
JAVA_DEBUGGER_PORT=29009
JMS_PROVIDER_PORT=27676
HTTP_LISTENER_PORT=28080
IIOP_SSL_LISTENER_PORT=23820
ASADMIN_LISTENER_PORT=24848
```

```
IIOP_SSL_MUTUALAUTH_PORT=23920
JMX_SYSTEM_CONNECTOR_PORT=28686
HTTP_SSL_LISTENER_PORT=28181
IIOP_LISTENER_PORT=23700

Successfully registered instance with DAS, now attempting to create Docker
container...
Command create-instance executed successfully.
```

When you invoke these commands, a Docker container of the name `DockerInstance1` will be created. If a Docker container of the same name already exists, the `create-instance` command will fail, and Payara Server will attempt to unregister the instance from the DAS.

WARNING	When "auto-naming" is enabled, Payara Server will only attempt to resolve conflicts with instance names - it will not attempt to resolve conflicts in container name.
WARNING	This type of instance is completely managed by the DAS - when you delete it, it will be deleted along with the Docker container including its logs (unless they are re-mapped to an external volume location)

From Docker

Payara Server supports creating instances by using Docker directly, making use of the "auto-naming" feature to resolve any conflicts.

The `docker container create` command creates a Payara Server Docker container without the need to contact the DAS. This is also the reason why the Docker container has a different name than the server instance - they have different contexts, and it is not possible to avoid naming conflicts, only to minimize them.

You can also specify an existing Docker node name in `PAYARA_NODE_NAME` system property (and ideally also `DOCKER_CONTAINER_IP`) and/or any relevant "deployment groups" by using the `PAYARA_DEPLOYMENT_GROUP` property. In this case, the Docker instance will be created as if it would be created from the DAS, except the fact that the DAS will be informed about that later when the container is properly started.

If the property was not specified or the node does not exist, a temporary Docker node and instance will be created on the DAS. The temporary Docker node is bound to the instance, so when you delete the instance, the temporary Docker node will be deleted too. This type of node is also not listed in the DAS or offered in inputs and cannot be edited.

CAUTION	Using a temporary Docker node and instance arrangement is not recommended for production scenarios.

If the instance running on a temporary Docker node is stopped from the DAS, it is immediately unregistered from the DAS. The next start of the same Docker container creates new a Payara Server Instance and a new temporary Docker node again despite the fact that it uses the same Docker container.

If the Docker containers are stopped externally (i.e. by the `docker` command), the node and instance will be marked for deletion and cleaned up on shutdown of the DAS.

Here is an example of how to create a Docker instance using this method:

```
docker container create --network host --mount type=bind,source="/app/passwordfile-
docker.txt",target="/pathInDocker/passwordfile.txt",readonly -e PAYARA_DAS_HOST
=payaraDas -e PAYARA_DAS_PORT=4848 -e PAYARA_PASSWORD_FILE
=/pathInDocker/passwordfile.txt payara/server-node:7.2.0
994e6d5db27630484348160193 2857fa224dfd9f9cda8578f3b09f8f11bf6004

docker start 994e6d5db276
994e6d5db276

docker logs 994e6d5db276
Docker Container ID is:
994e6d5db27630484348160193 2857fa224dfd9f9cda8578f3b09f8f11bf6004
No Docker container IP override given, setting to first result from 'hostname -I'
Hostname is 192.168.1.103
Docker Container IP is: 192.168.1.103
No Instance name given.
No node name given.
WARNING: Could not find a matching Docker Node: Creating a temporary node specific
to this container - cleanup of this container cannot be done by Payara Server
Creating a temporary node with an autogenerated name.
./payara7/bin/asadmin -I false -T -a -H payaraDas -p 4848 -W
/pathInDocker/passwordfile.txt _create-node-temp --nodehost 192.168.1.103
Running command create-local-instance:
./payara7/bin/asadmin -I false -T -a -H payaraDas -p 4848 -W
/pathInDocker/passwordfile.txt create-local-instance --node Sarcastic-Catfish
--dockernode true --ip 192.168.1.103
```

```
Setting Docker Container ID for instance Cooperative-Spookfish:
994e6d5db27630484348160193 2857fa224dfd9f9cda8578f3b09f8f11bf6004
./payara7/bin/asadmin -I false -H payaraDas -p 4848 -W
/pathInDocker/passwordfile.txt _set-docker-container-id --instance Cooperative-
Spookfish --id 994e6d5db27630484348160193 2857fa224dfd9f9cda8578f3b09f8f11bf6004
Command _set-docker-container-id executed successfully.
Starting instance Cooperative-Spookfish
...
```

IMPORTANT	Don't forget that you are configuring a connection to DAS from the viewpoint of the instance container, so ensure that the container can reach the DAS in the same network.

Starting and Stopping a Docker Instance

Instances on temporary Docker nodes have their lifecycle bound to a started container. So from the DAS point of view, either they are running or do not exist.

Starting a Docker instance on standard Docker node should be done just as if it were an instance on an SSH node. When the `asadmin start-instance` command is invoked, the DAS will contact the Docker Rest API as configured in the node config, and start the Docker container and the instance within it.

TIP	If the command remains unresponsive, the Docker instance probably failed to start. Use the `docker logs` command to see what happened.

```
asadmin --passwordfile /app/passwordfile.txt start-instance DockerInstance1
Command start-instance executed successfully.

asadmin --passwordfile /app/passwordfile.txt stop-instance DockerInstance1
The instance, DockerInstance1, is stopped.
Command stop-instance executed successfully.
```

Deleting a Docker Instance

Much as with when creating a standard Docker instance, deleting a Docker instance is done in the same way as other instances: with the `asadmin delete-instance` command. This will unregister the instance from the DAS, and delete the Docker container.

Containers on temporary Docker nodes are not deleted by the DAS, they will be only

stopped and removed from the DAS management including the temporary Docker node. The container management is controlled by the Docker.

```
asadmin --passwordfile /app/passwordfile.txt delete-instance DockerInstance1

Successfully removed instance DockerInstance1 from the DAS configuration, and
removed the container from node DockerNode1 (dockerHost1).
Command delete-instance executed successfully.
```

WARNING

if you would delete the container directly with the `docker` command, the DAS would not be aware of it. Such inconsistency can be resolved only by deletion of the instance from the DAS. This is automatically done on DAS restart, to guarantee an eventual consistency of the instances state on the system.

Configuring the Docker Container

Configuration of the Docker containers where server instances are hosted is done via system properties in the corresponding instance's configuration object (so can be shared across multiple instances that may belong to a deployment group).

A complete list of the available configuration options can be found in the Docker Engine REST API here:
link:https://docs.docker.com/engine/api/v1.39/#operation/ContainerCreate

IMPORTANT

The image name is not configurable. Payara Server expects the image name to match the value from the node configuration.

The configuration within Payara Server of the settings denoted in the above link takes the form of dotted names. These names adhere to the following syntax:

- Each property is prefixed with `Docker`
- Each child object is specified individually, with all of its parents prepended to it
- Arrays must be surrounded with square braces (`[]`)
- Array values are separated using the vertical bar symbol (`|`)

- The colon character (`;`) is used to denote the value of an object within an array
- Objects within an array are separated using a comma (`,`)

NOTE

Properties that are denoted by arrays of objects containing further objects or arrays are not currently supported. The `Env` property is unique in that the colon character is used to denote the equals sign, as Payara Server does not currently support properties that contain an equals in their value.

See below for some examples:

Example	Original JSON	Payara System Properties
Arrays must be surrounded with square braces & array values separated using the vertical bar symbol `|`	`{ENV: [arg1=foo,arg2=bar]}`	`Docker.Env=[arg1:foo|arg2:bar]`
Each child object of a parent object is specified individually	`{HostConfig: {Memory: 2048, CpuShares: 3}`	`Docker.HostConfig.Memory=2048` `Docker.HostConfig.CpuShares=3`
The colon character is used to denote the value of an object within an array & objects within an array are separated using a comma `,`	`{HostConfig: {BlkioDeviceReadBps: [{Path: /opt/foo, Rate: 24},{Path: /opt/bar, Rate: 48}]}`	`Docker.HostConfig.BlkioDeviceReadBps=[Path:/opt/foo,Rate:24|Path:/opt/bar,Rate:48]`

Reserved Environment Properties

The following Docker Environment properties are used by the default Docker image, `payara/server-node:7.2.0`, which you may wish to override to match your configuration (particularly if creating containers directly from Docker):

Environment Property	Use	Default Value
`PAYARA_DAS_HOST`	The IP address or hostname of the Domain Administration Server that the instance will register itself to.	`localhost`
`PAYARA_DAS_PORT`	The port that the Domain Administration Server is running on	`4848`
`PAYARA_NODE_NAME`	The name of the node that the instance should be created on.	""
`PAYARA_INSTANCE_NAME`	The name of the instance to be created.	""
`PAYARA_CONFIG_NAME`	The name of the config that the created instance should use.	""
`DOCKER_CONTAINER_IP`	The IP address or hostname that the Docker Container should use. This is used for verifying that a given node's network config maps to this container, or for when creating new nodes and instances.	First result of `hostname -I` (all IP addresses, excluding loopback)

Environment Property	Use	Default Value
`PAYARA_DEPLOYMENT_GROUP`	The name of the "Deployment Group" that the instance should join. Once the instance joins the Deployment Group, all the application targeted at the Deployment Group will automatically deploy to it.	""

IMPORTANT The DAS expects to be able to reach each instance using the port listed in its configuration. Don't forget to ensure that the DAS and containers reside on the same network.

Other Examples for Docker Instances

Creating a Container Using the Docker REST API

NOTE You can alternatively create a JSON file and then use curl syntax for sending files (i.e. `@create.json`).

```
curl -H 'Accept: application/json' -H 'Content-Type: application/json' -i
'http://dockerHost1:2376/containers/create?name=ManagedContainer2' --data '{
  "Image": "payara/server-node:mytag",
  "HostConfig": {
    "Mounts": [
      {
        "Type": "bind",
        "Source": "/app/passwordfile-docker.txt",
        "Target": "/opt/payara/passwords/passwordfile.txt",
        "ReadOnly": true
      }
    ],
    "NetworkMode": "host"
  },
  "Env": [
    "PAYARA_DAS_HOST=payaraDas",
    "PAYARA_DAS_PORT=4848",
    "PAYARA_NODE_NAME=DockerNode1",
    "PAYARA_INSTANCE_NAME=ManagedContainer2",
    "DOCKER_CONTAINER_IP=dockerHost1"
  ]
```

```
}'

HTTP/1.1 201 Created
Api-Version: 1.39
Content-Type: application/json
Docker-Experimental: false
Ostype: linux
Server: Docker/18.09.7 (linux)
Date: Mon, 04 Nov 2019 13:15:13 GMT
Content-Length: 90

{"Id":"e7803ce3ec964805c41d8a0eef5838299b5b8d38aa9e0801f05f3bc56b8d5fa1",
"Warnings":null}

>> curl -i 'http://dockerHost1:2376/containers/ManagedContainer2/start' --data ''
HTTP/1.1 204 No Content
Api-Version: 1.39
Docker-Experimental: false
Ostype: linux
Server: Docker/18.09.7 (linux)
Date: Mon, 04 Nov 2019 13:17:15 GMT
```

Creating a Container with a Set Instance Name, Resolving Conflicts

```
docker container create --network host --mount type=bind,source="/app/passwordfile-
docker.txt",target="/pathInDocker/passwordfile.txt",readonly -e PAYARA_DAS_HOST
=payaraDas -e PAYARA_DAS_PORT=4848 -e DOCKER_CONTAINER_IP=dockerHost1 -e
PAYARA_PASSWORD_FILE=/pathInDocker/passwordfile.txt -e PAYARA_NODE_NAME=DockerNode1
-e PAYARA_INSTANCE_NAME=ManagedContainer2 payara/server-node:mytag
af48bec58c144bad8ac83c9344dcebc4b9a6d528dd8673a6e6f5275e8b3ed2a2

docker start af48bec58c14
af48bec58c14

docker logs af48bec58c14
Docker Container ID is:
af48bec58c144bad8ac83c9344dcebc4b9a6d528dd8673a6e6f5275e8b3ed2a2
Docker Container IP is: dockerHost1
Instance name provided, but local file system for instance missing, checking if
file system or new instance needs to be created.
Checking if an instance with name ManagedContainer2 has been registered with the
DAS
./payara7/bin/asadmin -I false -t -H payaraDas -p 4848 -W
/pathInDocker/passwordfile.txt list-instances --nostatus ManagedContainer2
Found an instance with name ManagedContainer2 registered to the DAS, checking if
registered Docker Container ID matches this containers ID
./payara7/bin/asadmin -I false -t -H payaraDas -p 4848 -W
/pathInDocker/passwordfile.txt _get-docker-container-id --instance
ManagedContainer2
Registered Docker Container ID is:
e7803ce3ec964805c41d8a0eef5838299b5b8d38aa9e0801f05f3bc56b8d5fa1
Docker Container IDs do not match, creating a new instance.
Node name provided, checking if node details match this container.
Node with matching name found, checking node details.
Node Host of matching node is nodes.node.DockerNode1.node-host=dockerHost1
Node details match, no need to create a new node.
Running command create-local-instance:
./payara7/bin/asadmin -I false -T -a -H payaraDas -p 4848 -W
```

```
/pathInDocker/passwordfile.txt create-local-instance --node DockerNode1
--dockernode true --ip dockerHost1 ManagedContainer2
Setting Docker Container ID for instance ManagedContainer2-PerfectZiege:
af48bec58c144bad8ac83c9344dcebc4b9a6d528dd8673a6e6f5275e8b3ed2a2
./payara7/bin/asadmin -I false -H payaraDas -p 4848 -W
/pathInDocker/passwordfile.txt _set-docker-container-id --instance
ManagedContainer2-PerfectZiege --id
af48bec58c144bad8ac83c9344dcebc4b9a6d528dd8673a6e6f5275e8b3ed2a2
Command _set-docker-container-id executed successfully.
Starting instance ManagedContainer2-PerfectZiege
```

Listing Docker Containers

```
>> docker ps -a
CONTAINER ID        IMAGE                                COMMAND
CREATED             STATUS              PORTS               NAMES
af48bec58c14        payara/server-node:mytag   "/opt/payara/entrypo…"   3 minutes
ago       Up 2 minutes                              gentle_piranha
e7803ce3ec96        payara/server-node:mytag   "/opt/payara/entrypo…"   7 minutes
ago       Up 5 minutes                              ManagedContainer2
994e6d5db276        payara/server-node:mytag   "/opt/payara/entrypo…"   16 minutes
ago      Up 15 minutes                             musing_euclid

docker images payara/server-node
REPOSITORY           TAG                 IMAGE ID            CREATED
SIZE
payara/server-node   custom-7.2.0       386a996b3649        About an hour ago
511MB
payara/server-node   7.2.0              f5cf02e10dc2        4 weeks ago
525MB
payara/server-node   latest              f5cf02e10dc2        4 weeks ago
525MB
payara/server-node   mytag               f5cf02e10dc2        4 weeks ago
525MB

asadmin --passwordfile /app/appservers/passwordfile.txt list-nodes

localhost-domain1  CONFIG  localhost
DockerNode1  DOCKER  dockerHost1
Command list-nodes executed successfully.

asadmin --passwordfile /app/appservers/passwordfile.txt list-instances

Cooperative-Spookfish           running
ManagedContainer2               running
ManagedContainer2-PerfectZiege   running
Command list-instances executed successfully.
```

Resynchronizing Payara Server Instances and the DAS

Configuration data for a Payara Server instance is stored as follows:

- In the repository of the domain administration server (DAS)
- In a cache on the host that is local to the instance

The configuration data in these locations must be synchronized. The cache is synchronized in the following circumstances:

- Whenever an `asadmin` subcommand is run. For more information, see impact-of-configuration-changes" in Payara Server General Administration section.
- When a user uses the administration tools to start or restart an instance.

Default Synchronization for Files and Directories

The `--sync` option of the subcommands for starting an instance controls the type of synchronization between the DAS and the instance's files when the instance is started. You can use this option to override the default synchronization behavior for the files and directories of an instance. For more information, see to-resynchronize-an-instance-and-the-das-online.

On the DAS, the files and directories of an instance are stored in the `domain-dir` directory, where `domain-dir` is the directory in which a domain's configuration is stored.

The default synchronization behavior for the files and directories of an instance is as follows:

`applications`: This directory contains a subdirectory for each application that is deployed to the instance.

+ By default, only a change to an application's top-level directory within the application directory causes the DAS to synchronize that application's directory.

+ When the DAS resynchronizes the`applications` directory, all the application's files and all generated content that is related to the application are copied to the instance.

+ If a file below a top-level subdirectory is changed without a change to a file in the top-level subdirectory, full synchronization is required. In `normal` operation, files below the top-level subdirectories of these directories are not changed and such files should not be changed by users.

+ If an application is deployed and un-deployed, full synchronization is not necessary to update the instance with the change.

`config`: This directory contains configuration files for the entire domain.

+ By default, the DAS resynchronizes files that have been modified since the last resynchronization only if the `domain.xml` file in this directory has been modified.

+ If you add a file to the `config` directory of an instance, the file is deleted when the instance is resynchronized with the DAS. However, any file that you add to the `config` directory of the DAS is not deleted when instances and the DAS are resynchronized. By default, any file that you add to the `config` directory of the DAS is not resynchronized.

+ If you require any additional configuration files to be resynchronized, you must specify the files explicitly.

+ For more information, see to-resynchronize-additional-configuration-files.

`config/config-name`: This directory contains files that are to be shared by all instances that reference the named configuration config-name. A config-name directory exists for each named configuration in the configuration of the DAS.

+ Because the `config-name` directory contains the subdirectories `lib` and `docroot`, this directory might be very large. Therefore, by default, only a change to a file or a top-level subdirectory of config-name causes the DAS to resynchronize the config-name directory.

`config/domain.xml`: This file contains the DAS configuration for the domain to which the instance belongs.

+ By default, the DAS resynchronizes this file if it has been modified since the last resynchronization.

+ NOTE: A change to the `config/domain.xml` file is required to cause the DAS to resynchronize an instance's files.
If the `config/domain.xml` file has not changed since the last resynchronization, none

of the instance's files is resynchronized, even if some of these files are out of date in the cache.

`docroot`: This directory is the HTTP document root directory. By default, all instances in a domain use the same document root directory.

+ To enable instances to use a different document root directory, a virtual server must be created in which the `docroot` property is set.

+ For more information, see the "`create-virtual-server`" help page.

+ The `docroot` directory might be very large. Therefore, by default, only a change to a file or a subdirectory in the top level of the `docroot` directory causes the DAS to resynchronize the `docroot` directory. The DAS checks files in the top level of the `docroot` directory to ensure that changes to the `index.html` file are detected.

+ When the DAS resynchronizes the `docroot` directory, all modified files and subdirectories at any level are copied to the instance.

+ If a file below a top-level subdirectory is changed without a change to a file in the top-level subdirectory, full synchronization is required.

`generated`: This directory contains generated files for Jakarta EE applications and modules, for example, EJB stubs, compiled JSP classes, and security policy files.

+ WARNING: Do not modify the contents of this directory.

+ This directory is resynchronized when the `applications` directory is resynchronized. Therefore, only directories for applications that are deployed to the instance are resynchronized.

`lib`: `lib/classes`: These directories contain common Jakarta EE class files or JAR archives and ZIP archives for use by applications that are deployed to the entire domain. Typically, these directories contain common JDBC drivers and other utility libraries that are shared by all applications in the domain.

+ The contents of these directories are loaded by the common class loader. For more information, see ""Using the Common Class Loader"" in the Payara Server Application Development section.

+ The class loader loads the contents of these directories in the following order:

+ . `lib/classes` . `lib/*.jar` . `lib/*.zip`

`lib/applibs`: This directory contains application-specific Jakarta class files or JAR archives and ZIP archives for use by applications that are deployed to the entire domain.

`lib/ext`: This directory contains optional packages in JAR archives and ZIP archives for use by applications that are deployed to the entire domain.

+ In past versions, these archive files are loaded by using Jakarta extension mechanism. Since Java 11+, as extension library support has been removed, these libraries will be automatically added as classpath elements.

The `lib` directory and its subdirectories typically contain only a small number of files. Therefore, by default, a change to any file in these directories causes the DAS to resynchronize the file that changed.

To Resynchronize an Instance and the DAS Online

Resynchronizing an instance and the DAS updates the instance with changes to the instance's configuration files on the DAS. An instance is resynchronized with the DAS when the instance is started or restarted.

NOTE	Resynchronization of an instance is only required if the instance is stopped. A running instance does not require resynchronization.

1. Ensure that the DAS is running.
2. Determine whether the instance is stopped.

```
> asadmin list-instances instance-name
```

instance-name: The name of the instance that you are resynchronizing with the DAS.

If the instance is stopped, the `list-instances` subcommand indicates that the instance is not running.

3. If the instance is stopped, start the instance.

   If the instance is running, no further action is required.

   If you require full synchronization, set the `--sync` option of the `start-instance` subcommand to `full`. If default synchronization is sufficient, omit this option.

```
> asadmin start-instance [--sync full] instance-name
```

   If SSH is set up, start the instance locally from the host where the instance resides.

   If you require full synchronization, set the `--sync` option of the `start-local-instance` subcommand to `full`. If default synchronization is sufficient, omit this option.

```
> asadmin start-local-instance [--node node-name] [--sync full] instance-name
```

This example determines that the instance `yml-i1` is stopped and fully resynchronizes the instance with the DAS. Because SSH is set up, the instance is started locally on the host where the instance resides.

In this example, multiple nodes are defined for the Payara Server installation that is running on the node's host.

To determine whether the instance is stopped, the following command is run in multimode on the DAS host:

```
asadmin> list-instances yml-i1
yml-i1   not running
Command list-instances executed successfully.
```

To start the instance, the following command is run in single mode on the host where

the instance resides:

```
> asadmin start-local-instance --node sj01 --sync full yml-i1
Removing all cached state for instance yml-i1.
Waiting for yml-i1 to start ..............
Successfully started the instance: yml-i1
instance Location: /export/payara7/glassfish/nodes/sj01/yml-i1
Log File: /export/payara7/glassfish/nodes/sj01/yml-i1/logs/server.log
Admin Port: 24849
Command start-local-instance executed successfully.
```

See Also

- "`list-instances`"
- "`start-instance`"
- "`start-local-instance`"

To Resynchronize Library Files

To ensure that library files are resynchronized correctly, you must ensure that each library file is placed in the correct directory for the type of file.

1. Place each library file in the correct location for the type of library file as shown in the following table.

Type of Library Files	Location
Common JAR archives and ZIP archives for all applications in a domain.	`domain-dir/lib`
Common Jakarta EE class files for a domain for all applications in a domain.	`domain-dir/lib/classes`
Application-specific libraries.	`domain-dir/lib/applibs`
Optional packages for all applications in a domain.	`domain-dir/lib/ext`
Library files for all applications that are deployed to a specific cluster or standalone instance.	`domain-dir/config/<config-name>/lib`

Type of Library Files	Location
Optional packages for all applications that are deployed to a specific cluster or standalone instance.	`domain-dir/config/<config-name>/lib/ext`

domain-dir: The directory in which the domain's configuration is stored. **config-name**: For a standalone instance: the named configuration that the instance references. For a clustered instance: the named configuration that the cluster to which the instance belongs references.

1. When you deploy an application that depends on these library files, use the `--libraries` option of the `deploy` subcommand to specify these dependencies.

   For library files in the `domain-dir/lib/applib` directory, only the JAR file name is required, for example:

   ```
   asadmin> deploy --libraries commons-coll.jar,X1.jar app.ear
   ```

   For other types of library file, the full path is required.

See Also

"`deploy`"

You can also view the full syntax and options of the subcommands by typing the command `asadmin help deploy` at the command line.

To Resynchronize Custom Configuration Files for an Instance

Configuration files in the `domain-dir/config` directory that are resynchronized for the entire domain.

If you create a custom configuration file for an instance or a cluster, the custom file is resynchronized only for the instance or cluster. You'll have to update any JVM options that reference this configuration file in the domain configuration settings.

1. Place the custom configuration file in the `domain-dir/config/config-name` directory. **config-name**: The named configuration that the instance references.
2. If the instance locates the file through an option of the Jakarta EE application launcher, update the option.
3. Delete the JVM option in case of a customized config file location.

   ```
   asadmin> delete-jvm-options --target instance-name option-name=current-value
   ```

   instance-name: The name of the instance for which the custom configuration file is created. **option-name**: The name of the option for locating the file. **current-value**: The current value of the option for locating the file.

4. Re-create the JVM option that you deleted in the previous step.

   ```
   asadmin> create-jvm-options --target instance-name option-name=new-value
   ```

   instance-name: The name of the instance for which the custom configuration file is created. **option-name**: The name of the option for locating the file. **new-value**: The new value of the option for locating the file.

This example updates the option for locating the `server.policy` file to specify a custom file location for the instance `pmd`.

```
asadmin> delete-jvm-options --target pmd -Djava.security.policy=${
com.sun.aas.instanceRoot}/config/server.policy
Deleted 1 option(s)
Command delete-jvm-options executed successfully.

asadmin create-jvm-options --target pmd -Djava.security.policy=${
com.sun.aas.instanceRoot}/config/pmd-config/server.policy
Created 1 option(s)
Command create-jvm-options executed successfully.
```

See Also

- "`create-jvm-options`"
- "`delete-jvm-options`"

To Resynchronize Users' Changes to Files

A change to the `config/domain.xml` file is required to cause the DAS to resynchronize instances' files. If other files in the domain directory are changed without a change to the `config/domain.xml` file, instances are not resynchronized with these changes.

The following changes are examples of changes to the domain directory without a change to the `config/domain.xml` file:

- Adding files to the `lib` directory
- Adding certificates to the key store by using the `keytool` command

1. Change the last modified time of the `config/domain.xml` file.

   Exactly how to change the last modified time depends on the operating system. For example, on UNIX and Linux systems, you can use `touch` command.

2. Resynchronize each instance in the domain with the DAS.

   For instructions, see to-resynchronize-an-instance-and-the-das-online.

See Also

- to-resynchronize-an-instance-and-the-das-online
- `touch`(1)

To Resynchronize Additional Configuration Files

By default, Payara Server synchronizes only the following configuration files:

- `admin-keyfile`
- `cacerts.jks`
- `default-web.xml`
- `domain.xml`
- `domain-passwords`

- `keyfile`
- `keystore.jks`
- `server.policy`
- `sun-acc.xml`
- `wss-server-config-1.0`
- `xml wss-server-config-2.0.xml`

If you require instances in a domain to be resynchronized with additional configuration files for the domain, you can specify a list of files to resynchronize.

CAUTION	If you specify a list of files to resynchronize, you must specify all the files that the instances require, including the files that Payara Server resynchronizes by default. Any file in the instance's cache that is not in the list is deleted when the instance is resynchronized with the DAS.

In the `config` directory of the domain, create a plain text file that is named `config-files` that lists the files to resynchronize.

In the `config-files` file, list each file name on a separate line.

This example shows the content of a `config-files` file. This file specifies that the `some-other-info` file is to be resynchronized in addition to the files that Payara Server resynchronizes by default:

```
admin-keyfile
cacerts.jks
default-web.xml
domain.xml
domain-passwords
keyfile
keystore.jks
server.policy
sun-acc.xml
wss-server-config-1.0.xml
wss-server-config-2.0.xml
some-other-info
```

To Prevent Deletion of Application-Generated Files

When the DAS resynchronizes an instance's files, the DAS deletes from the instance's cache any files that are not listed for resynchronization.

If an application creates files in a directory that the DAS resynchronizes, these files are deleted when the DAS resynchronizes an instance with the DAS.

Put the files in a subdirectory under the domain directory that is not a pre-defined directory, for example, `/payara7/glassfish/domains/domain1/myapp/myfile`.

To Resynchronize an Instance and the DAS Offline

Resynchronizing an instance and the DAS offline updates the instance's cache without the need for the instance to be able to communicate with the DAS. Offline resynchronization is typically required for the following reasons:

- To reestablish the instance after an upgrade
- To synchronize the instance manually with the DAS when the instance cannot contact the DAS

Follow these steps:

1. Ensure that the DAS is running.
2. Export the configuration data that you are resynchronizing to an archive file.

NOTE	Only the options that are required to complete this task are provided in this step. For information about all the options for exporting the configuration data, see the "`export-sync-bundle`" help page.

   How to export the data depends on the host from where you run the `export-sync-bundle` subcommand.

   - From the DAS host, run the `export-sync-bundle` subcommand as follows:

     ```
     asadmin> export-sync-bundle --target target
     ```

target: The cluster or standalone instance for which to export configuration data.

Do not specify a clustered instance. If you specify a clustered instance, an error occurs. To export configuration data for a clustered instance, specify the name of the cluster of which the instance is a member, not the instance.

The file is created on the DAS host.

- From the host where the instance resides, run the `export-sync-bundle` subcommand as follows:

```
> asadmin --host das-host [--port admin-port] export-sync-bundle [--retrieve
=true] --target target
```

das-host: The name of the host where the DAS is running. **admin-port**: The HTTP or HTTPS port on which the DAS listens for administration requests. If the DAS listens on the default port for administration requests, you may omit this option. **target**: The cluster or standalone instance for which to export configuration data.

Do not specify a clustered instance. If you specify a clustered instance, an error occurs. To export configuration data for a clustered instance, specify the name of the cluster of which the instance is a member, not the instance.

NOTE	To create the archive file on the host where the instance resides, set the `--retrieve` option to `true`. If you omit this option, the archive file is created on the DAS host.

3. If necessary, copy the archive file that you created in **Step 2** from the DAS host to the host where the instance resides.
4. From the host where the instance resides, import the instance's configuration data from the archive file that you created in **Step 2**.

NOTE	Only the options that are required to complete this task are provided in this step. For information about all the options for importing the

> configuration data, see the "`import-sync-bundle`" help page.

```
> asadmin import-sync-bundle [--node node-name] --instance instance-name
archive-file
```

node-name: The node on which the instance resides. If you omit this option, the subcommand determines the node from the DAS configuration in the archive file. **instance-name**: The name of the instance that you are resynchronizing. **archive-file**: The name of the file, including the path, that contains the archive file to import.

This example resynchronizes the clustered instance `yml-i1` and the DAS offline. The instance is a member of the cluster `ymlcluster`. The archive file that contains the instance's configuration data is created on the host where the instance resides.

```
asadmin --host dashost.example.com export-sync-bundle --retrieve=true --target
ymlcluster
Command export-sync-bundle executed successfully.

asadmin import-sync-bundle --node sj01 --instance yml-i1 ymlcluster-sync-bundle.zip
Command import-sync-bundle executed successfully.
```

See Also

- "`export-sync-bundle`"
- "`import-sync-bundle`"

The Extra Terse Option

The `extraterse` option is intended for use with scripts, adding an extra level of terseness to the command output to the CLI.

> **NOTE** Currently, this feature only works with the `create-instance` and `create-local-instance` commands.

When enabled, the `create-instance` and `create-local-instance` commands should only return the name of the instance created.

The intention behind this feature being that you should be able to set variables from this

output for use with scripts, since you can't know beforehand what the name of an instance is if a name conflict is resolved or a name is generated from scratch.

This option can be enabled either by specifying `--extraterse` or simply `-T` on the command line (in comparison, the normal terse option is enabled with `--terse` or `-t`) before or after the command name. Being a CLI option rather than a command parameter, it is recommended that you specify it before the command name, as the ability to specify asadmin options (like `--host`) after the command name is technically deprecated syntax and will be ignored in certain circumstances.

IMPORTANT	The `extraterse` option implicitly enables the terse option - this is true for **all** commands (not just `create-instance` and `create-local-instance`).

Extraterse Examples

```
$INSTANCE_NAME = asadmin create-local-instance --node localhost-domain1 --autoname
--extraterse
asadmin start-instance $INSTANCE_NAME

$SECOND_INSTANCE_NAME = asadmin -a -T create-local-instance --node localhost-
domain1
```

Migrating EJB Timers

If a Payara Server instance stops or fails abnormally, it may be desirable to migrate the EJB timers defined for that stopped server instance to another running server instance.

Automatic timer migration is enabled by default for clustered server instances that are stopped normally. Automatic timer migration can also be enabled to handle clustered server instance crashes.

In addition, timers can be migrated manually for stopped or crashed server instances.

- to-enable-automatic-ejb-timer-migration-for-failed-clustered-instances
- to-migrate-ejb-timers-manually

To Enable Automatic EJB Timer Migration for Failed Clustered Instances

Automatic migration of EJB timers is enabled by default for clustered server instances that are stopped normally. The procedure in this section is only necessary if you want to enable automatic timer migration for clustered server instances that have stopped abnormally.

NOTE	If the GMS is enabled, the default automatic timer migration cannot be disabled. To disable automatic timer migration, you must first disable the GMS. For information about the GMS, see group-management-service.

Before You Begin

Automatic EJB timer migration can only be configured for clustered server instances. Automatic timer migration is not possible for standalone server instances.

Enable delegated transaction recovery for the cluster.

This enables automatic timer migration for failed server instances in the cluster.

For instructions on enabling delegated transaction recovery, see "administering-transactions" in Payara Server General Administration section.

To Migrate EJB Timers Manually

EJB timers can be migrated manually from a stopped source instance to a specified target instance in the same cluster if GMS notification is not enabled.

If no target instance is specified, the DAS will attempt to find a suitable server instance. A migration notification will then be sent to the selected target server instance.

Note the following restrictions:

- If the source instance is part of a cluster, then the target instance must also be part of that same cluster.
- It is not possible to migrate timers from a standalone instance to a clustered instance, or from one cluster to another cluster.

- It is not possible to migrate timers from one standalone instance to another standalone instance.
- All EJB timers defined for a given instance are migrated with this procedure. It is not possible to migrate individual timers.

Before You Begin

The server instance from which the EJB timers are to be migrated should not be active during the migration process.

1. Verify that the source clustered server instance from which the EJB timers are to be migrated is not currently running.

```
> asadmin list-instances source-instance
```

2. Stop the instance from which the timers are to be migrated, if that instance is still running.

```
> asadmin stop-instance source-instance
```

> **NOTE** The target instance to which the timers will be migrated should be running.

3. List the currently defined EJB timers on the source instance, if desired.

```
> asadmin list-timers source-cluster
```

4. Migrate the timers from the stopped source instance to the target instance.

```
> asadmin migrate-timers --target target-instance source-instance
```

The following example show how to migrate timers from a clustered source instance named `football` to a clustered target instance named `soccer`.

```
asadmin> migrate-timers --target soccer football
```

See Also

"`list-timers`", "`migrate-timers`", "`list-instances`", "`stop-instance`"

Administering Named Configurations

A named configuration is a set of configuration information for Payara Server instances and clusters. A configuration sets port numbers for listeners that an instance or a cluster uses and defines settings for items such as the EJB container, security, logging, and monitoring. Applications and resources are not defined in named configurations.

Payara Server enables you to create configurations for use by clusters and instances. Payara Server creates configurations automatically for clusters and instances that do not reference an existing configuration. You can modify any existing named configuration regardless of how the configuration was created.

About Named Configurations

Configurations exist in a domain. Multiple Payara Server instances or clusters in the domain can reference the same configuration, or they can have separate configurations. To ensure that the environment in a cluster's instances is homogenous, all instances in the cluster inherit the cluster's configuration.

Types of Named Configurations

Each named configuration is one of the following types of configuration:

Standalone configuration: A standalone configuration is referenced by only one instance or cluster and is not shared with any other instances or clusters.

+ NOTE: A configuration that is referenced by only one cluster is a standalone configuration, even if the cluster contains multiple instances.

Shared configuration: A shared configuration is referenced by multiple instances (even if they live on a deployment group) or clusters.

The type of un-clustered instances is determined by the type of the configuration that the instance references.

For more information, see types-of-payara-server-instances.

The `default-config` Configuration

The `default-config` configuration is a special configuration that acts as a template for creating new named configurations. Clusters and instances cannot refer to the `default-config` configuration. The `default-config` configuration can only be copied to create configurations.

Automatically Created Configurations

When you create a cluster or an instance, you can choose whether to specify an existing configuration that the new cluster or instance will reference. If you choose to create a cluster or an instance without specifying an existing configuration, Payara Server automatically creates a configuration for the cluster or instance.

For more information, see the following sections:

- to-create-a-cluster
- to-create-an-instance-centrally
- to-create-an-instance-locally

Payara Server automatically creates a configuration by copying the `default-config` configuration. If you require an instance or cluster to reference a copy of a different configuration, copy the configuration and specify the copy when you create the instance or cluster.

For information about how to copy a configuration, see to-create-a-named-configuration.

Payara Server assigns the name `<cluster-or-instance>-config` to an automatically created configuration. `<cluster-or-instance>` is the name of the cluster or instance for which the configuration is created.

The `server-config` configuration is automatically created for the domain administration server (DAS) when the domain is created.

Directory for Configuration Synchronization

When a named configuration is created, Payara Server creates a configuration directory on the domain administration server (DAS) at `domain-dir/config/config-name`.

domain-dir: The directory in which the domain's configuration is stored. **config-name**: The name that was assigned to the configuration when the configuration was created.

This contents of this directory are synchronized to all instances that inherit or reference the configuration.

Creating, Listing, and Deleting Named Configurations

Payara Server enables you to create configurations for use by clusters and instances, obtain information about configurations, and delete configurations that are no longer required.

To Create a Named Configuration

Use the `copy-config` subcommand in remote mode to create a named configuration by copying an existing configuration. Payara Server requires you to create a configuration by copying a configuration because a configuration contains many required settings.

The newly created configuration is identical to the configuration that you copy until you change its configuration settings.

TIP	Create a named configuration only if you plan to share the configuration among multiple un-clustered instances or clusters.

Otherwise, create clusters and instances without specifying an existing configuration. If no configuration is specified, Payara Server creates a copy of the default configuration for the cluster or instance.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the `copy-config` subcommand.

NOTE	Only the options that are required to complete this task are provided in this step. For information about all the options for configuring the named configuration, see the copy-config help page.

```
asadmin> copy-config source-config-name destination-config-name
```

source-config-name: The name of the configuration that you are copying. You must specify a configuration to copy even if you are copying the default configuration. The configuration must already exist. **destination-config-name**: Your choice of name for the configuration that you are creating by copying the source configuration.

The following example creates the named configuration `cluster-presets-config` by copying the default configuration.

```
asadmin> copy-config default-config cluster-presets-config
Command copy-config executed successfully.
```

See Also

- to-create-a-cluster
- to-create-an-instance-centrally
- to-create-an-instance-locally
- copy-config

You can also view the full syntax and options of the subcommand by typing `asadmin help copy-config` at the command line.

To List the Named Configurations in a Domain

Use the `list-configs` subcommand in remote mode to list existing named configurations in a domain.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Run the "`list-configs`" subcommand.

```
asadmin> list-configs
```

The following example lists all named configurations in the current domain.

```
asadmin> list-configs
server-config
default-config
cluster-presets-config
Command list-configs executed successfully.
```

See Also

"`list-configs`"

You can also view the full syntax and options of the subcommand by typing `asadmin help list-configs` at the command line.

To List the Targets of a Named Configuration

Use the `list-clusters` subcommand and the `list-instances` subcommand in remote mode to list the targets of a named configuration.

The targets of a named configuration are the clusters and Payara Server instances that reference the configuration.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. List the clusters that refer to the configuration.

```
asadmin> list-clusters config-name
```

config-name: The name of the configuration whose targets you are listing.

3. List the instances that refer to the configuration.

```
asadmin> list-instances config-name
```

config-name: The name of the configuration whose targets you are listing.

The following example shows that the cluster `ymlcluster` and the instances `yml-i1` and `yml-i2` reference the named configuration `ymlcluster-config`.

```
asadmin> list-clusters ymlcluster-config

ymlcluster partially running
Command list-clusters executed successfully.
asadmin> list-instances ymlcluster-config
yml-i1   running
yml-i2   not running
Command list-instances executed successfully.
```

See Also

- "`list-clusters`"
- "`list-instances`"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help list-clusters`
- `asadmin help list-instances`

To Delete a Named Configuration

Use the `delete-config` subcommand in remote mode to delete an existing named configuration from the configuration of the DAS.

You cannot delete the `default-config` configuration.

NOTE	A standalone configuration that was created automatically for a cluster or a Payara Server instance is deleted when the cluster or instance is deleted.

Before You Begin

Ensure that no clusters or instances refer to the configuration. If a cluster or instance

refers to the configuration and is no longer required, delete the cluster or instance.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. Confirm that no clusters refer to the configuration that you are deleting.

   ```
   asadmin> list-clusters config-name
   ```

   config-name: The name of the configuration that you are deleting.

3. Confirm that no instances refer to the configuration that you are deleting.

   ```
   asadmin> list-instances config-name
   ```

   config-name: The name of the configuration that you are deleting.

4. Run the "`delete-config`" subcommand.

   ```
   asadmin> delete-config config-name
   ```

   config-name: The name of the configuration that you are deleting.

The following example confirms that no clusters or instances refer to the configuration `cluster-presets-config` and then deletes the configuration.

```
asadmin> list-clusters cluster-presets-config
Nothing to list
Command list-clusters executed successfully.

asadmin> list-instances cluster-presets-config
Nothing to list.
Command list-instances executed successfully.

asadmin> delete-config cluster-presets-config
Command delete-config executed successfully.
```

See Also

- to-delete-an-instance-centrally
- to-delete-an-instance-locally

- to-delete-a-cluster
- delete-config
- list-clusters
- list-instances

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help delete-config`
- `asadmin help list-clusters`
- `asadmin help list-instances`

Modifying Properties for Named Configurations and Instances

The properties in a named configuration define port numbers for un-clustered instances that reference the configuration or clustered instances that inherit the configuration. An instance initially obtains port numbers from the configuration that the instance references or inherits.

CAUTION	To avoid port conflicts, edit the properties of named configurations and instances.

Properties for Port Numbers in a Named Configuration

The default configuration `default-config` contains properties that define the initial values of port numbers in a configuration that is copied from `default-config`. When an instance or a cluster that references the configuration is created, these properties are set for the instance.

You can create additional system properties for a configuration either by specifying the `--systemproperties` option of the "`copy-config`" subcommand or by using the "`create-system-properties`" subcommand.

To reference a system property from the configuration, use the `${property-name}`

notation, where prop-name is the name of the system property.

For example, if a configuration defines additional HTTP listeners, use system properties to define the ports for those listeners. However, these properties are not set automatically when an instance or a cluster that references the configuration is created.

NOTE	You must set these properties explicitly when you create the instance or cluster.

The properties in a named configuration that define port numbers are as follows:

`ASADMIN_LISTENER_PORT`: This property specifies the port number of the HTTP port or HTTPS port through which the DAS connects to the instance to manage the instance. `HTTP_LISTENER_PORT`: This property specifies the port number of the port that is used to listen for HTTP requests. `HTTP_SSL_LISTENER_PORT`: This property specifies the port number of the port that is used to listen for HTTPS requests. `IIOP_LISTENER_PORT`: This property specifies the port number of the port that is used for IIOP connections. `IIOP_SSL_LISTENER_PORT`: This property specifies the port number of the port that is used for secure IIOP connections. `IIOP_SSL_MUTUALAUTH_PORT`: This property specifies the port number of the port that is used for secure IIOP connections with client authentication. `JAVA_DEBUGGER_PORT`: This property specifies the port number of the port that is used for connections to the Java Platform Debugger Architecture (JPDA) debugger. `JMS_PROVIDER_PORT`: This property specifies the port number for the Jakarta Message Service provider. `JMX_SYSTEM_CONNECTOR_PORT`: This property specifies the port number on which the JMX connector listens. `OSGI_SHELL_TELNET_PORT`: This property specifies the port number of the port that is used for connections to the Apache Felix Remote Shell.

+ This shell uses the Felix shell service to interact with the OSGi module management subsystem.

TIP	Valid port values are `1-65535`.

WARNING	On UNIX operating systems, creating sockets that listen on ports `1-1024` requires superuser privileges.

To Modify a Named Configuration's Properties

Use the `get` subcommand and the `set` subcommand in remote mode to modify a named configuration's properties.

You might copy a configuration for use by instances that reside on the same host as instances that refer to the original configuration. In this situation, edit the properties of one of the configurations to ensure that instances that will refer to the configuration have the correct initial settings.

If you change the port number in a configuration, the port number is changed for any instance that references or inherits the configuration.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. For each property that you are modifying, determine the current value and set the new value.
3. Determine the current value of the property.

```
asadmin> get configs.config.config-name.system-property.property-name.value
```

config-name: The name of the configuration whose properties you are modifying.
property-name: The name of the property that you are modifying. For a list of available properties, see properties-for-port-numbers-in-a-named-configuration.

4. Set the property to its new value.

```
asadmin> set configs.config.config-name.system-property.property-name.value=new-value
```

config-name: The name of the configuration whose properties you are modifying.
property-name: The name of the property that you are modifying. For a list of available properties, see properties-for-port-numbers-in-a-named-configuration. **new-**

value: The value to which you are setting the property.

The following example changes the value of the `JMS_PROVIDER_PORT` property in the `cluster-presets-config` configuration from `27676` to `27678`.

```
asadmin> get configs.config.cluster-presets-config.system-
property.JMS_PROVIDER_PORT.value

configs.config.cluster-presets-config.system-property.JMS_PROVIDER_PORT.value=27676
Command get executed successfully.

asadmin> set configs.config.cluster-presets-config.system-
property.JMS_PROVIDER_PORT.value=27678

Command set executed successfully.
```

See Also

- "`get` command"
- "`set` command"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help get`
- `asadmin help set`

To Modify Port Numbers of an Instance

Use the `get` subcommand and the `set` subcommand in remote mode to modify the port numbers of an instance.

The port numbers of an instance are initially set in the configuration that the instance references or inherits from its parent cluster. Multiple instances that reside on the same host must each listen on a unique port number.

Therefore, if multiple instances that reference or inherit the same configuration reside on the same host, a port conflict prevents all except one of the instances from starting. To avoid port conflicts, modify the port numbers on which individual instances listen.

If you modify an instance's port number and later modify the port number in the instance's configuration, the instance's port number remains unchanged.

The port numbers of an instance are stored as Java system properties. When Payara Server is started, it treats these properties in the same way as properties that are passed through the `-D` option of the Java application launcher.

1. Ensure that the DAS is running. Remote subcommands require a running server.
2. For each port number that you are modifying, determine the current value and set the new value.
3. Determine the current value of the port number.

   ```
   asadmin> get
   servers.server.instance-name.system-property.port-property.value
   ```

   instance-name: The name of the instance whose port numbers you are modifying. **port-property**: The name of the property that corresponds to the port number that you are modifying. For a list of available properties, see properties-for-port-numbers-in-a-named-configuration.

4. Set the port number to its new value.

   ```
   asadmin> get
   servers.server.instance-name.system-property.port-property.value=new-value
   ```

   instance-name: The name of the instance whose port numbers you are modifying. **port-property**: The name of the property that corresponds to the port number that you are modifying. For a list of available properties, see properties-for-port-numbers-in-a-named-configuration. **new-value**: The value to which you are setting the port number.

The following example changes the port number of the HTTP port or the HTTPS port for administration of the `pmdsainst` instance from `24849` to `24859`.

```
asadmin> get servers.server.pmdsainst.system-property.ASADMIN_LISTENER_PORT.value

servers.server.pmdsainst.system-property.ASADMIN_LISTENER_PORT.value=24849
```

```
Command get executed successfully.
asadmin> set servers.server.pmdsainst.system-property.ASADMIN_LISTENER_PORT.value
=24859

Command set executed successfully.
```

See Also

- "`get` command"
- "`set` command"

You can also view the full syntax and options of the subcommands by typing the following commands at the command line:

- `asadmin help get`
- `asadmin help set`

Configuring HTTP Load Balancing

This chapter describes how to configure HTTP load balancing on Payara Server.

Setting up HTTP Load Balancing

This section describes how to set up load balancing for Payara Server.

Prerequisites for Setting up HTTP Load Balancing

Before configuring your load balancer, you must:

- Install a supported web server (or web proxy) and configure it. If using the `mod_jk` or `mod_proxy_ajp` modules, the only supported web server is Apache HTTP Server 2.x.
- Configure the `mod_jk` connector module, as described in configuring-payara-server-with-apache-http-server-and-mod_jk, or configure the `mod_proxy_ajp` connector module, as described in configuring-payara-server-with-apache-http-server-and-mod_proxy_ajp.
- Create Payara Server clusters or server instances (in a deployment group) to participate in load balancing.

- Deploy applications to these clusters or instances.

Configuring Payara Server with Apache HTTP Server and `mod_jk`

Payara Server can be configured for load balancing with Apache HTTP Server as a front end by enabling the Apache `mod_jk` connector module. To enable the `mod_jk` module in Payara Server, set the Payara Server `jk-enabled network-listener` attribute.

You can also create `jk-connectors` under different virtual-servers using the `jk-enabled network-listener` attribute.

To Configure the `mod_jk` Connector Module

1. Install Apache HTTP Server and mod_jk.
2. Configure `workers.properties` and `httpd.conf`. For example:
   - `apache2/config/workers.properties`

```
# Define 1 real worker using ajp13
worker.list=worker1
# Set properties for worker1 (ajp13)
worker.worker1.type=ajp13
worker.worker1.host=localhost
worker.worker1.port=8009
```

   - `apache2/conf/httpd.conf`

```
LoadModule jk_module /Users/Amy/apache2/modules/mod_jk-1.2.25-httpd-2.2.4.so
JkWorkersFile /Users/Amy/apache2/conf/worker.properties
# Where to put jk logs
JkLogFile /Users/Amy/apache2/logs/mod_jk.log
# Set the jk log level [debug/error/info]
JkLogLevel debug
# Select the log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
# JkOptions indicate to send SSL KEY SIZE,
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories
# JkRequestLogFormat set the request format
JkRequestLogFormat "%w %V %T"
# Send everything for context /examples to worker named worker1 (ajp13)
JkMount /examples/* worker1
```

3. Start the Apache HTTP Server.
4. Create a `jk-enabled` network listener by using the "`create-network-listener`"

subcommand.

```
asadmin> create-network-listener --protocol http-listener-1 --listenerport 8009
--jkenabled true jk-connector
```

5. If you are using an external file (like `payara-jk.properties`), set the `jk-configuration-file` property of the network listener to its absolute path's location.

```
asadmin> set server-config.network-config.network-listeners.network-listener.jk-
connector.jk-configuration-file=domain-dir/config/payara-jk.properties
```

6. If you expect to need more than five threads for the listener, increase the maximum threads in the `http-thread-pool` pool:

```
asadmin> set configs.config.server-config.thread-pools.thread-pool.http-thread-
pool.max-thread-pool-size=value
```

7. Restart the Payara Server instances.

This example shows an `httpd.conf` file that is set for load balancing.

```
LoadModule jk_module /usr/lib/httpd/modules/mod_jk.so
JkWorkersFile /etc/httpd/conf/worker.properties
# Where to put jk logs
JkLogFile /var/log/httpd/mod_jk.log
# Set the jk log level [debug/error/info]
JkLogLevel debug
# Select the log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
# JkOptions indicate to send SSL KEY SIZE,
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories
# JkRequestLogFormat set the request format
JkRequestLogFormat "%w %V %T"
# Send all jsp requests to Payara Server
JkMount /*.jsp worker1
# Send all glassfish-test requests to Payara Server
JkMount /payara-test/* loadbalancer
```

The following example shows a `workers.properties` or `payara-server-jk.properties` file that is set for load balancing. The `worker.worker*.port` should match with the JK ports you configured:

```
worker.list=worker1,worker2,loadbalancer
worker.worker1.type=ajp13
worker.worker1.host=localhost
worker.worker1.port=8009
worker.worker1.lbfactor=1
worker.worker1.socket_keepalive=1
worker.worker1.socket_timeout=300
worker.worker2.type=ajp13
worker.worker2.host=localhost
worker.worker2.port=8010
worker.worker2.lbfactor=1
worker.worker2.socket_keepalive=1
worker.worker2.socket_timeout=300
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=worker1,worker2
```

Configuring Payara Server with Apache HTTP Server and `mod_proxy_ajp`

Payara Server can be configured for load balancing with Apache HTTP Server as a front end by enabling the Apache `mod_proxy_ajp` connector module.

To enable the `mod_proxy_ajp` module in Payara Server, set the Payara Server `jk-enabled network-listener` attribute. You can also create `jk-connectors` under different virtual-servers using the `jk-enabled network-listener` attribute.

To Configure the `mod_proxy_ajp` Connector Module

1. Install Apache HTTP Server.
2. Configure the `httpd.conf` configuration file. For example:

```
LoadModule proxy_module /usr/lib/httpd/modules/mod_proxy.so
LoadModule proxy_ajp_module /usr/lib/httpd/modules/mod_proxy_ajp.so

Listen 1979
NameVirtualHost *:1979
<VirtualHost *:1979>
   ServerName localhost
   ErrorLog /var/log/apache2/ajp.error.log
   CustomLog /var/log/apache2/ajp.log combined

   <Proxy *>
     AddDefaultCharset Off
     Order deny,allow
     Allow from all
   </Proxy>

   ProxyPass / ajp://localhost:8009/
   ProxyPassReverse / ajp://localhost:8009/
</VirtualHost>
```

3. Start the Apache HTTP Server.
4. Create a `jk-enabled` network listener by using the `create-network-listener` subcommand.

```
asadmin> create-network-listener --protocol http-listener-1 --listenerport 8009
--jkenabled true jk-connector
```

5. If you expect to need more than five threads for the listener, increase the maximum threads in the `http-thread-pool` pool:

```
asadmin> set configs.config.server-config.thread-pools.thread-pool.http-thread-
pool.max-thread-pool-size=value
```

6. Restart the Payara Server instances.

HTTP Load Balancer Deployments

You can configure your load balancer in different ways, depending on your goals and environment, as described in the following sections:

Using Clustered Server Instances

The most common way to deploy the load balancer is with a cluster or clusters of server instances. By default, all the instances in a cluster have the same configuration and the same applications deployed to them. The load balancer distributes the workload between the server instances and requests fail over from an unhealthy instance to a healthy one. If you've configured HTTP session persistence, session information persists when the request is failed over.

If you have multiple clusters, requests can be load balanced across clusters but are only failed over between the instances in a single cluster. Use multiple clusters in a load balancer to easily enable **rolling upgrades** of applications.

Using Multiple Standalone Instances

It is also possible to configure your load balancer to use multiple standalone instances, and load balance and failover requests between them. However, in this configuration,

you must manually ensure that the standalone instances have homogenous environments and the same applications deployed to them.

Because clusters automatically maintain a homogenous environment, for most situations it is better and easier to use clusters.

TIP	Load balancing across multiple standalone instances only provides failover for requests, and any associated HTTP session data will not be failed over. This is another reason why using a cluster, which can provide session failover, is a more desirable load balancing configuration.

Configuring High Availability Session Persistence and Failover

This chapter explains how to enable and configure high availability session persistence.

Overview of Session Persistence and Failover

Payara Server provides high availability session persistence through failover of HTTP session data and stateful session bean (SFSB) session data. Failover means that in the event of a server instance or hardware failure, another server instance in a Cluster takes over a distributed session.

For example, Jakarta EE applications typically have significant amounts of session state data. A web shopping cart is the classic example of session state. Also, an application can cache frequently-needed data in the session object. In fact, almost all applications with significant user interactions need to maintain session state.

NOTE	When using high availability session persistence together with a load balancer, use a load balancer that includes session-based stickiness as part of its load-balancing algorithm. Otherwise, session data can be misdirected or lost.

Requirements

A distributed session can run in multiple Payara Server instances, if:

- Each server instance has access to the same session state data.

  Payara Server supports in-memory session replication on standalone server instances (and as part of a deployment group) through the use of the Data Grid.

  In-memory session replication is enabled by default for Payara Server traditional clusters if the **Group Management Service** is enabled. For more information about GMS, see group-management-service.

  If server instances in a cluster are located on different hosts, ensure that the following prerequisites are met:

  - To ensure that GMS and in-memory replication function correctly, the hosts must be on the same subnet.
  - To ensure that in-memory replication functions correctly, the system clocks on all hosts in the cluster must be synchronized as closely as possible.
- Each server instance has the same distributable web application deployed to it. The `web-app` element of the `web.xml` deployment descriptor file must contain the `distributable` element.
- The web application uses high-availability session persistence. If a non-distributable web application is configured to use high-availability session persistence, the server writes an error to the log file.
- The web application must be deployed using the `deploy` subcommand with the `--availabilityenabled` option set to `true`. For more information on these subcommands, see deploy.

Restrictions

When configuring session persistence and failover, note the following restrictions:

- When a session fails over, any references to open files or network connections are lost. Applications must be coded with this restriction in mind.
- EJB Singletons are created for each server instance in a cluster, and not once per

cluster.

- The high availability session persistence service is not compatible with dynamic deployment, dynamic reloading, and auto-deployment features.

  These features are for development, not production environments, so you must disable them before enabling the session persistence service.

  For information about how to disable these features, see the "Payara Server Application Deployment section".

- Payara Server does not support High Availability Database (HA-DB) configurations. Instead, use in-memory replication, as described in high-availability-session-persistence.
- You can only bind certain objects to distributed sessions that support failover. Contrary to the Jakarta Servlet specification, Payara Server does not throw an `IllegalArgumentException` if an object type not supported for failover is bound into a distributed session.

  You can bind the following objects into a distributed session that supports failover:

  - Local home and object references for all EJB components.
  - Co-located stateless session, stateful session, or entity bean reference .
  - Distributed stateless session, stateful session, or entity bean reference.
  - JNDI Context for `InitialContext` and `java:comp/env`.
  - `UserTransaction` objects.

    However, if the instance that fails is never restarted, any prepared global transactions are lost and might not be correctly rolled back or committed.

  - Serializable Jakarta EE object types.
- You cannot bind the following object types into sessions that support failover:
  - JDBC DataSource

- Jakarta Messaging `ConnectionFactory` and `Destination` objects
- Jakarta Mail Session
- Connection Factory
- Administered Objects
- Web service reference

  In general, for these objects, failover will not work. However, failover might work in some cases, if for example the object is serializable.

Scope

The availability service can be enabled for the following scopes, ranging from highest to lowest:

- Cluster
- Standalone server instance (may be part of a deployment group)
- Web, EJB, or JMS container in a cluster
- Application
- Standalone Web, EJB, or JMS module
- Individual Stateful Session Bean (SFSB)

In general, enabling or disabling availability session persistence for a cluster or container involves setting the boolean `availability-service` property to `true` or `false` by means of the `asadmin set` subcommand.

The availability service is enabled by default for Payara Server clusters and all Web, EJB, and JMS containers running in a cluster.

The value set for the `availability-service` property is inherited by all child objects running in a given cluster or container unless the value is explicitly overridden at the individual module or application level.

For example, if the `availability-service` property is set to `true` for an EJB

container, the availability service will be enabled by default for all EJB modules running in that container.

Conversely, to enable availability at a given scope, you must enable it at all higher levels as well. For example, to enable availability at the application level, you must also enable it at the cluster or server instance and container levels.

Enabling the High Availability Session Persistence Service

This section explains how to configure and enable the high availability session persistence service.

To Enable Availability for a Cluster, Standalone Instance or Container

This procedure explains how to enable high availability for a cluster as a whole, or for Web, EJB, or JMS containers that run in a cluster, or for a standalone instance that may belong to a deployment group.

1. Create a Payara Server cluster or a new deployment group.
2. Set up load balancing for the cluster/deployment group.

   For instructions, see setting-up-http-load-balancing.

3. Verify that all instances within the cluster/deployment group for which you want to enable availability are running.

   These steps are also necessary when enabling availability for a Web, EJB, or JMS container.

   a. Verify that all instances in the cluster/deployment group are running.

   ```
   asadmin> list-instances
   ```

   A list of instances and their status is displayed. If the instances for which you want to enable availability are not running, you can start them by using the following command for each instance:

```
asadmin> start-instance instance-name
```

4. Use one of the following `asadmin set` subcommands to enable availability for a specific cluster or deployment group configuration, or for a specific Web, EJB, or JMS container.
   - For a cluster/deployment group as a whole

     ```
     asadmin> set <config>-name.availability-service.availability-enabled=true
     ```

     For example, for a cluster named `c1`:

     ```
     asadmin> set c1-config.availability-service.availability-enabled=true
     ```

   - For the Web container in a cluster/deployment group

     ```
     asadmin> set cluster-config.availability-service.web-container-
     availability.availability-enabled=true
     ```

   - For the EJB container in a cluster/deployment group

     ```
     asadmin> set cluster-config.availability-service.ejb-container-
     availability.availability-enabled=true
     ```

   - For the JMS container in a cluster/deployment group

     ```
     asadmin> set cluster-config.availability-service.jms-
     availability.availability-enabled=true
     ```

   - For a standalone server instance:

     ```
     asadmin> set <instance-name>-config.availability-service.availability-
     enabled=true
     ```

5. Restart the standalone server instance or each server instance in the cluster/deployment group.

6. Enable availability for any SFSB that requires it.

   Select methods for which checkpointing the session state is necessary.

   For more information, see configuring-availability-for-an-individual-bean.

7. Make each web module distributable if you want it to be highly available.

   For more information, see "web-module-deployment-guidelines" in the Payara Server Application Deployment section.

8. Enable availability for individual applications, web modules, or EJB modules during deployment.

   See the links below for instructions.

See Also

- configuring-availability-for-individual-web-applications
- using-single-sign-on-with-session-failover

Configuring Availability for Individual Web Applications

To enable and configure availability for an individual web application, edit the application deployment descriptor file (`glassfish-web.xml` or `payara-web.xml`). The settings in an application's deployment descriptor override the web container's availability settings.

The `session-manager` element's `persistence-type` attribute determines the type of session persistence an application uses.

It must be set to `hazelcast` (as this is the default persistence mode) to enable high availability session persistence.

Example

```
<payara-web-app>
  <session-config>
    <session-manager persistence-type="hazelcast">
```

```
      <manager-properties>
        <property name="persistenceFrequency" value="web-method" />
      </manager-properties>
      <store-properties>
        <property name="persistenceScope" value="session" />
      </store-properties>
    </session-manager>
  </session-config>
</payara-web-app>
```

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

Configuring Datagrid Replication and Multi-Threaded Concurrent Access to `HttpSessions`

If you are using data replication via the Data Grid (`hazelcast`) and your web application involves multiple client threads concurrently accessing the same session ID, then you may experience session loss even without any instance failure.

The problem is that the Payara Server Data Grid's replication framework makes use of session versioning.

This feature was designed with the more traditional HTTP request/response communication model in mind. However, for some types of applications, the traditional request/response model does not work. Examples include many Ajax-related frameworks and the use of **Frames**.

Another example is when a page includes many static resources, such as `JPG` files. In these situations, most browsers will optimize the loading of these resources by using multiple parallel connections, each of which is handled by a separate request processing thread. If the application has already established a session, then this will also involve more than one thread at a time accessing a single `HttpSession`.

The solution in such cases is to use the `relaxVersionSemantics` property in the `glassfish-web.xml` or `payara-web.xml` deployment descriptor file for the application.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a

> future release. It is recommended to use payara-web.xml instead.

This enables the web container to return for each requesting thread whatever version of the session that is in the active cache regardless of the version number.

This is critical when multiple threads are interacting in an essentially non-deterministic fashion with the container.

Example

The following is an example snippet from a `payara-web.xml` file that illustrates where to add the `relaxVersionSemantics` property.

```
<payara-web-app>
  <session-config>
    <session-manager persistence-type="hazelcast">
      <manager-properties>
        <property name="relaxCacheVersionSemantics" value="true"/>
      </manager-properties>
    </session-manager>
  </session-config>
</payara-web-app>
```

Using Single Sign-On with Session Failover

In a single application server instance, once a user is authenticated by an application, the user is not required to re-authenticate individually to other applications running on the same instance. This is called single sign-on (SSO).

For this feature to continue to work even when an HTTP session fails over to another instance in a cluster or deployment group, single sign-on information must be persisted using in-memory replication via the Data Grid

To persist single sign-on information, first, enable availability for the server instance and the web container, and then enable single-sign-on state failover.

You can enable single sign-on state failover by using the `asadmin set` command to set the configuration's `availability-service.web-container-availability.sso-failover-enabled` property to true.

For example, use the `set` command as follows, where `config1` is the configuration name:

```
asadmin> set config-1.availability-service.web-container-availability.sso-failover-enabled="true"
```

Single Sign-On Groups

Applications that can be accessed through a single name and password combination constitute a **single sign-on group**.

For HTTP sessions corresponding to applications that are part of a single sign-on group, if one of the sessions times out, other sessions are not invalidated and continue to be available. This is because time out of one session should not affect the availability of other sessions.

As a corollary of this behavior, if a session times out and you try to access the corresponding application from the same browser window that was running the session, you are not required to authenticate again. However, a new session is created.

Take the example of a shopping cart application that is a part of a single sign-on group with two other applications. Assume that the session time out value for the other two applications is higher than the session time out value for the shopping cart application.

If your session for the shopping cart application times out and you try to run the shopping cart application from the same browser window that was running the session, you are not required to authenticate again. However, the previous shopping cart is lost, and you have to create a new shopping cart.

The other two applications continue to run as usual even though the session running the shopping cart application has timed out.

Similarly, suppose a session corresponding to any of the other two applications times out. You are not required to authenticate again while connecting to the application from the same browser window in which you were running the session.

NOTE

This behavior applies only to cases where the session times out.
If single sign-on is enabled and you invalidate one of the sessions using `HttpSession.invalidate()`, the sessions for all applications belonging to the single sign-on group are invalidated

If you try to access any application belonging to the single sign-on group, you are required to authenticate again, and a new session is created for the client accessing the application.

Stateful Session Bean Failover

Stateful session beans (SFSBs) contain client-specific state. There is a one-to-one relationship between clients and the stateful session beans. At creation, the EJB container gives each SFSB a unique session ID that binds it to a client.

An SFSB state can be saved in a persistent store in case a server instance fails. The state of an SFSB is saved to the persistent store at predefined points in its life cycle. This is called *checkpointing*. If enabled, checkpointing generally occurs after the bean completes any transaction, even if the transaction rolls back.

However, if an SFSB participates in a bean-managed transaction, the transaction might be committed in the middle of the execution of a bean method. Since the bean's state might be undergoing transition as a result of the method invocation, this is not an appropriate time to checkpoint the bean's state.

In this case, the EJB container checkpoints the bean's state at the end of the corresponding method, provided the bean is not in the scope of another transaction when that method ends. If a bean-managed transaction spans across multiple methods, checkpointing is delayed until there is no active transaction at the end of a subsequent method.

The state of an SFSB is not necessarily transactional and might be significantly modified as a result of non-transactional business methods. If this is the case for an SFSB, you can specify a list of checkpointed methods, as described in specifying-

methods-to-be-checkpointed

If a distributable web application references an SFSB, and the web application's session fails over, the EJB reference is also failed over.

If an SFSB that uses session persistence is un-deployed while the Payara Server instance is stopped, the session data in the persistence store might not be cleared.

To prevent this, undeploy the SFSB while the Payara Server instance is running.

Configuring Availability for the EJB Container

To enable availability for the EJB container use the `asadmin set` command to set the following three properties for the configuration:

- `availability-service.ejb-container-availability.availability-enabled`
- `availability-service.ejb-container-availability.sfsb-persistence-type`
- `availability-service.ejb-container-availability.sfsb-ha-persistence-type`

For example, if `config1` is the configuration name, use the following commands:

```
asadmin> set config1.availability-service.ejb-container-availability.availability-enabled="true"

asadmin> set config1.availability-service.ejb-container-availability.sfsb-persistence-type="file"

asadmin> set config1.availability-service.ejb-container-availability.sfsb-ha-persistence-type="hazelcast"
```

Configuring the SFSB Session Store When Availability Is Disabled

If availability is disabled, the local file system is used for SFSB state passivation, but not persistence. To change where the SFSB state is stored, change the **Session Store Location** setting in the EJB container.

Configuring Availability for an Individual Application or EJB Module

You can enable SFSB availability for an individual application or EJB module during deployment:

- If you are deploying with the Administration Console, check the *Availability Enabled* checkbox.
- If you are deploying using use the `asadmin deploy` commands, set the `--availabilityenabled` option to `true`.

  For more information, see "`deploy`".

Configuring Availability for an Individual Bean

To enable availability and select methods to be checkpointed for an individual SFSB, use the `glassfish-ejb-jar.xml` deployment descriptor file. To enable high availability session persistence, set `availability-enabled=true` in the `ejb` element, like this:

```
<glassfish-ejb-jar>
    <enterprise-beans>
        <ejb availability-enabled="true">
            <ejb-name>MySFSB</ejb-name>
        </ejb>
    </enterprise-beans>
</glassfish-ejb-jar>
```

Specifying Methods to Be Checkpointed

If enabled, checkpointing generally occurs after the bean completes any transaction, even if the transaction rolls back.

To specify additional optional checkpointing of SFSBs at the end of non-transactional business methods that cause important modifications to the bean's state, use the `checkpoint-at-end-of-method` element in the `ejb` element of the `glassfish-ejb-jar.xml` deployment descriptor file.

The non-transactional methods in the `checkpoint-at-end-of-method` element can be:

- `create()` methods defined in the home interface of the SFSB, if you want to checkpoint the initial state of the SFSB immediately after creation
- For SFSBs using container managed transactions only, methods in the remote interface of the bean marked with the transaction attribute `TX_NOT_SUPPORTED` or `TX_NEVER`
- For SFSBs using bean managed transactions only, methods in which a bean managed transaction is neither started nor committed

  Any other methods mentioned in this list are ignored.

At the end of invocation of each of these methods, the EJB container saves the state of the SFSB to persistent store.

NOTE

If an SFSB does not participate in any transaction, and if none of its methods are explicitly specified in the `checkpoint-at-end-of-method` element, the bean's state is not checkpointed at all even if `availability-enabled="true"` for this bean. For better performance, specify a small subset of methods. The methods should accomplish a significant amount of work or result in important modification to the bean's state.

Example of EJB Deployment Descriptor Specifying Method Checkpoints

```
<glassfish-ejb-jar>
    <enterprise-beans>
        <ejb availability-enabled="true">
            <ejb-name>ShoppingCartEJB</ejb-name>
            <checkpoint-at-end-of-method>
                <method>
                    <method-name>addToCart</method-name>
                </method>
            </checkpoint-at-end-of-method>
        </ejb>
    </enterprise-beans>
</glassfish-ejb-jar>
```

Configuring Java Message Service High Availability

This chapter describes how to configure the high availability features of the Java

Message Service (JMS). It covers how to configure Message Queue broker clusters and how to use them to provide connection failover and load balancing.

NOTE	It is important to be aware of the distinction between a message broker cluster (a group of broker instances) and a Payara Server cluster (a traditional group of server instances) to fully understand the concepts and instructions described in this chapter.

Documentation for the *Open Message Queue* software can be found in the following online resources:

- OpenMQ Technical Overview
- OpenMQ Administration Guide

Using Message Queue Broker Clusters with Payara Server

This section describes how the JMS service uses Message Queue broker clusters to support **high-availability** JMS messaging in Payara Server clusters and deployment groups. It describes the different cluster and broker types that are supported and how to configure them.

About Message Queue Broker Clusters

The following discussion provides a brief overview of Message Queue broker clusters.

Message Queue supports two clustering models both of which provide a scalable message service, but with each providing a different level of message service availability:

Conventional broker clusters: A conventional broker cluster provides for service availability. When a broker fails, clients connected to the failed broker reconnect to another broker in the cluster.

+ However, messages and state information stored in the failed broker cannot be recovered until the failed broker is brought back online.

+ The broker failure can therefore result in a significant delay and in JMS message order semantics not being preserved.

+ Message Queue supports two types of conventional cluster, based on where the cluster configuration change record is stored:

- **Conventional cluster with master broker**: In a conventional cluster with a master broker, one of the brokers, designated as the **master broker**, stores and maintains the cluster configuration change record. The other brokers in the cluster must communicate with the master broker to keep abreast of changes to the cluster configuration.

  This is the simplest broker cluster to configure, and is the type of broker cluster that Payara Server uses by default.

- **Conventional cluster of peer brokers**: In a conventional cluster of peer brokers, the cluster configuration change record is stored in a **JDBC data store** accessible to all the brokers.

  Thus, brokers can access cluster configuration information whether any other brokers in the cluster are running or not.

Enhanced broker clusters: An enhanced broker cluster provides for data availability in addition to service availability. When a broker fails, another broker takes over the pending work of the failed broker.

+ The failover broker has access to the failed broker's messages and state information. Clients connected to the failed broker reconnect to the failover broker.

+ In an enhanced cluster, as compared to a conventional cluster, messages owned by the failed broker are delivered by the failover broker **as soon as it takes over**, and JMS message order semantics are preserved.

+ IMPORTANT: By its very nature, an enhanced broker cluster is always cluster of peer brokers.

NOTE	Despite the message service availability offered by both conventional and enhanced broker clusters, they do not provide a guarantee against failure and the possibility that certain failures, for example in the middle of a transaction, could require that some operations be repeated. It is the responsibility of the messaging application (both producers and consumers) to respond to JMS exceptions appropriately.

Configuring Payara Server Clusters to Use Message Queue Broker Clusters

When a Payara Server cluster is created, the JMS service automatically configures a Message Queue **conventional broker cluster with master broker** for the cluster, provided that the JMS host type in the Payara Server cluster's configuration is either `Embedded` or `Local`.

The JMS service configures one Message Queue broker for each instance in the Payara Server cluster, and designates as master broker the broker associated with the **first instance** created in the cluster.

- In the case of `Embedded` JMS hosts, each broker inherently runs on the same host as the instance with which it is associated because it runs in the same JVM as the instance.
- In the case of `Local` JMS hosts, the JMS service configures each broker to run on the same host as the instance with which it is associated.

The JMS service manages the lifecycle of `Embedded` and `Local` JMS hosts, and this management extends to the management of Message Queue broker clusters as `Embedded` and `Local` JMS hosts. For a Payara Server cluster whose configuration specifies `Embedded` or `Local` JMS host type, the JMS service:

1. Creates and manages ONE Message Queue broker for each instance in the Payara Server cluster, using this broker instance as the primary JMS host for the instance, keeping a one-to-one relationship between both.
2. Maintains the JMS host list for each instance in the Payara cluster such that its

primary JMS host appears first in its JMS host list.

The JMS service supports the following types of Message Queue broker clusters with Payara Server clusters, based on the JMS host type:

`Embedded`: * Conventional broker cluster with master broker (default) * Conventional broker cluster of peer brokers `Local`: * Conventional broker cluster with master broker (default) * Conventional broker cluster of peer brokers * Enhanced broker cluster `Remote`: * Conventional broker cluster with master broker.

+ Brokers can differ in number from Payara Server instances and can be located on other hosts * Conventional broker cluster of peer brokers.

+ Brokers can differ in number from Payara Server instances and can be located on other hosts * Enhanced broker cluster.

+ Brokers can differ in number from Payara Server instances and can be located on other hosts.

IMPORTANT	It is recommended that in a robust production environment configuration, the JMS service must be configured in either `Local` or `Remote` mode. The `Embedded` mode is not recommended for the usual reliability and scalability that a production system may require.

The following topics provide instructions for configuring broker clusters in all these contexts.

Deployment Groups Considerations

The instructions provided will also apply to **deployment groups** as well, however, it is mandatory that all Payara Server instances reference the same named configuration object. If multiple instances in the same deployment group reference multiple, separate configuration objects, high availability for the JMS service won't be guaranteed.

When running the commands listed in the respective instructions, instead of targeting a

server cluster, you can target a deployment group instead.

To Configure a Payara Cluster to Use an Embedded or Local Conventional Broker Cluster with Master Broker

Use the `configure-jms-cluster` subcommand in remote `asadmin` mode to configure a conventional broker cluster with master broker to service a Payara Server cluster that uses either `Embedded` or `Local` JMS hosts.

Note that the `Embedded` mode brokers configuration is the default for Payara Server clusters.

Before You Begin: Perform the following steps after you have created the Payara Server cluster, but before you have added instances to the cluster or started the cluster.

CAUTION	Before using this procedure to reconfigure an existing cluster, you must follow the special procedures to migrate to another type of broker cluster, as described in to-migrate-between-types-of-embedded-or-local-conventional-broker-clusters. Failing to perform these special procedures could lead to data loss or corruption and even render your setup unusable, depending on the JMS operations performed on the existing cluster.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. Configure the Payara Server cluster to use a Message Queue conventional broker cluster with master broker by using the "`configure-jms-cluster`" subcommand:

```
asadmin configure-jms-cluster --clustertype=conventional --configstoretype
=masterbroker payara-cluster-name
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help configure-jms-cluster` at the command line.

To Configure a Payara Server Cluster to Use an Embedded or Local Conventional Broker Cluster of Peer Brokers

Use the `configure-jms-cluster` subcommand in remote `asadmin` mode to configure a conventional broker cluster of peer brokers to service a Payara Server cluster that uses `Embedded` or `Local` JMS hosts.

Before You Begin: Perform the following steps after you have created the Payara Server cluster, but before you have added instances to the cluster or started it.

CAUTION	Before using this procedure to reconfigure an existing cluster, you must follow the special procedures to migrate to another type of broker cluster, as described in to-migrate-between-types-of-embedded-or-local-conventional-broker-clusters. Failing to perform these special procedures could lead to data loss or corruption and even render your setup unusable, depending on the JMS operations performed on the existing cluster.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. Create a password file with the entry `AS_ADMIN_JMSDBPASSWORD` specifying the password of the database user.

   For information about password file entries, see the "`asadmin`" help page.
3. Place a copy of, or a link to, the database's JDBC driver `.jar` file in the appropriate directory, depending on the JMS host type, on each host where a Payara Server cluster instance is to run:
   - Embedded: `as-install-parent/payara/lib/install/applications/jmsra`
   - Local: `as-install-parent/mq/lib/ext`
4. Configure the Payara Server cluster to use a Message Queue conventional broker cluster with master broker by using the "`configure-jms-cluster`" subcommand:

```
asadmin --passwordfile password-file configure-jms-cluster --clustertype
=conventional --configstoretype=shareddb --dbvendor database-vendor-name
--dbuser database-user-name --dburl database-url --property list-of-database-
specific-properties <server-cluster-name>
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help configure-jms-cluster` at the command line.

To Change the Master Broker in an Embedded or Local Broker Cluster

Use the `change-master-broker` subcommand in remote `asadmin` mode to change the master broker to a different broker in a conventional broker cluster with master broker serving a Payara Server cluster that uses `Embedded` or `Local` JMS hosts.

Follow this procedure, for example, before you remove from a Payara Server cluster the instance associated with the current master broker.

Before You Begin: Although not an absolute requirement, you should make sure all Payara Server instances and Message Queue brokers in the cluster are running before using the `change-master-broker` command in order to avoid later internal configuration synchronization of any unavailable instance or broker.

1. Ensure that the server is running.

   Remote `asadmin` subcommands require a running server.

2. Change the master broker by using the "`change-master-broker`" subcommand:

```
asadmin change-master-broker payara-clustered-instance-name
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help change-master-broker` at the command line.

To Migrate between Types of Embedded or Local Conventional Broker Clusters

If the need arises to convert from a conventional broker cluster with master broker to a conventional broker cluster of peer brokers, or the reverse, follow the instructions in the **Managing Conventional Clusters** section of the OpenMQ Administration Guide.

To Configure a Payara Server Cluster to Use a Local Enhanced Broker Cluster

Use the `configure-jms-cluster` subcommand in remote `asadmin` mode to configure an enhanced broker cluster to service a Payara Server cluster that uses `Local` JMS hosts.

Before You Begin: Perform the following steps after you have created the Payara Server cluster, but before you have added instances to the cluster or started it.

CAUTION	Before using this procedure to reconfigure an existing cluster, you must follow the special procedures to migrate from a conventional broker cluster to an enhanced broker cluster, as described in the **Converting a Conventional Cluster to an Enhanced Cluster** section in the OpenMQ Administration Guide. Failing to perform these special procedures could lead to data loss or corruption and even render your setup unusable, depending on the JMS operations performed on the existing cluster.

1. Ensure that the server is running. Remote `asadmin` subcommands require a running server.
2. Create a password file with the entry `AS_ADMIN_JMSDBPASSWORD` specifying the password of the database user.

   For information about password file entries, see the "`asadmin`" help page.
3. Place a copy of, or a link to, the database's JDBC driver `.jar` file in the `as-install-parent/mq/lib/ext` directory on each host where a Payara Server cluster instance is running.

4. Configure the Payara Server cluster to use a Message Queue enhanced broker cluster by using the "`configure-jms-cluster`" subcommand:

```
asadmin --passwordfile password-file configure-jms-cluster --clustertype
=enhanced --configstoretype=shareddb --messagestoretype=jdbc --dbvendor
database-vendor-name --dbuser database-user-name --dburl database-url --property
list-of-database-specific-properties payara-cluster-name
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help configure-jms-cluster` at the command line.

To Configure a Payara Server Cluster to Use a Remote Broker Cluster

Before You Begin

Perform the following steps after you have:

- Used Message Queue to create a broker cluster.
- Created the Payara Server cluster, but not yet created instances for the cluster.

1. Ensure that the server is running. The remote subcommands used in this procedure require a running server.
2. Delete the `default_JMS_host` JMS host by using the "`delete-jms-host`" subcommand:

```
asadmin delete-jms-host --target payara-cluster-name default_JMS_host
```

3. Create a JMS host for each broker in the broker cluster by using the "`create-jms-host`" subcommand.

   For each broker, use an `asadmin create-jms-host` command instruction like this:

```
asadmin create-jms-host --target payara-cluster-name --mqhost broker-host
--mqport broker-port --mquser mq-user --mqpassword mq-user-password <jms-
host-name-for-broker>
```

4. Start the brokers in the cluster by using the Message Queue `imqbrokerd` command, as described in the **Managing Broker Clusters** section the in OpenMQ Administration Guide.
5. Create instances in the Payara Server cluster, as described in to-create-an-instance-centrally and to-create-an-instance-locally.

Connection Failover

The use of Message Queue broker clusters provides JMS connection failover, including several options that control how connection failures are handled.

Use the Administration Console's Java Message Service page to configure these options. To display this page, click the configuration for the Payara server cluster or instance's configuration in the navigation pane, and then click the **Java Message Service** link on the Configuration page.

The way in which connection failover operates depends on whether the broker cluster is configured to be conventional or enhanced:

- In a **conventional cluster**, when a broker fails, clients may reconnect to any other broker in the cluster.

  The *Reconnect* field specifies whether reconnection should take place, and the *Address List Behavior* and *Address List Iterations* fields specify how the client chooses what broker to fail over to.

- In an **enhanced cluster**, when a broker fails, another broker automatically takes over its messages and clients.

  Clients automatically fail over to the appropriate broker. The *Reconnect*, *Address List Behavior* and *Address List Iterations* fields are ignored.

For more information on connection failover, including how failover on conventional clusters differs from failover on enhanced clusters, see the **Automatic Reconnection** in the OpenMQ Administration Guide.

Reconnect: Applies only to conventional clusters. Enables reconnection and connection failover.

+ When **disabled**, the Java Message Service does not attempt to reconnect if a connection fails. **Reconnect Interval**: Specifies the number of seconds between reconnection attempts.

+ If it is too short, this time interval does not give a broker time to recover.

+ If it is too long, the wait time might represent an unacceptable delay.

+ The default value is `5` seconds. **Reconnect Attempts**: Specifies the number of attempts to connect (or reconnect) to a particular JMS host before trying another host in the JMS host list. The host list is also known as the *Address List*.

+ TIP: Hosts are chosen from the address list either in order or randomly, depending on the setting of Address List Behavior. **Address List Behavior**: For conventional clusters, this field specifies how the Java Message Service selects which JMS host in the JMS hosts list to initially connect to, and if the broker fails, how the Java Message Service selects which JMS host in the JMS hosts list to fail over to.

+ For enhanced clusters, this field specifies how the Java Message Service selects which JMS host in the JMS hosts list to initially connect to.

+ When performing initial connection or, for conventional clusters only, when performing failover, then if this attribute is set to `Priority`, the Java Message Service tries to connect to the first JMS host specified in the JMS hosts list and uses another one only if the first one is not available. If this attribute is set to `Random`, the Java Message Service selects the JMS host randomly from the JMS hosts list. If that host is not available, another one is chosen randomly.

+ The default value for `Embedded` and `Local` JMS host types is `Priority`, and the default for the Remote JMS host type is `Random`.

+ For `Embedded` and `Local` JMS host types, the Java Message Service ensures that

the Message Queue broker servicing a clustered instance appears first in that instance's JMS host list.

+ Thus, having `Priority` as the default *Address List Behavior* ensures that an application deployed to a clustered instance will always try to create its initial connection to that instance's co-located broker.

+ TIP: If there are many clients attempting a connection using the same connection factory, use the `Random` setting to prevent them from all attempting to create their initial connection to the same JMS host.

Address List Iterations: For conventional clusters, this field specifies the number of times the Java Message Service iterates through the JMS hosts list in an effort to establish its initial connection.

+ If the broker fails, this field specifies the number of times the Java Message Service iterates through the JMS hosts list in an effort to fail over to another broker.

+ For enhanced clusters, this field specifies the number of times the Java Message Service iterates through the JMS hosts list in an effort to establish its initial connection. If the broker fails, this field is not used when performing reconnection.

You can override these settings using JMS connection factory settings. For details, see "administering-jms-connection-factories-and-destinations" in the Payara Server General Administration section.

Load-Balanced Delivery to Mdbs

When a message-driven bean (MDB) application is deployed to a Payara Server cluster, incoming messages are delivered randomly to MDBs without regard to the server cluster instances in which they are running.

If the MDB is configured to receive messages from a durable or non-durable subscription on a topic, then only one MDB instance across the whole Payara Server cluster will receive each message.

For more information about these features, see the **About Shared Topic Subscriptions for Clustered Containers** section in the OpenMQ Administration Guide.

RMI-IIOP Load Balancing and Failover

This chapter describes using high-availability features for remote EJB references and JNDI objects over RMI-IIOP in Payara Server.

- "Overview"
- "InitialContext Load Balancing"
- "Per-Request Load Balancing (PRLB)"

Overview

With RMI-IIOP load balancing, IIOP client requests are distributed to different server instances or name servers. The goal is to spread the load evenly across the cluster, thus providing scalability. IIOP load balancing combined with EJB clustering and availability also provides EJB failover.

General Requirements for Configuring Load Balancing

Payara Server provides high availability of remote EJB references and `NameService` objects over RMI-IIOP, provided all the following apply:

- Your deployment has a cluster (or deployment group) of at least two instances.
- Jakarta EE applications are deployed to all instances and clusters that participate in load balancing.
- RMI-IIOP client applications are enabled for load balancing.

Payara Server supports load balancing for Jakarta applications executing in the Application Client Container (ACC).

See enabling-rmi-iiop-hardware-load-balancing-and-failover.

CAUTION	Payara Server does not support RMI-IIOP load balancing and failover over secure sockets layer (SSL).

Load Balancing Models

Payara Server supports two general models for load balancing:

initialcontext-load-balancing

When a client performs a JNDI lookup for an object, the Naming Service creates a `InitialContext` (IC) object associated with a particular server instance.

From then on, all lookup requests made using that IC object are sent to the same server instance. `InitialContext` load balancing can be configured automatically across an entire cluster.

per-request-load-balancing-prlb

Per Request Load Balancing (PRLB) is a method for load balancing stateless EJBs that enables load-balancing for each request to an EJB instance.

NOTE	PRLB chooses the first node in a cluster to use on each request. PRLB is configured on a per-EJB basis.

`InitialContext` Load Balancing

`InitialContext` Summary

When `InitialContext` load balancing is used, the client calls the `InitialContext()` method to create a new `InitialContext` (IC) object that is associated with a particular server instance. JNDI lookups are then performed on that IC object, and all lookup requests made using that IC object are sent to the same server instance.

All `EJBHome` objects looked up with that `InitialContext` are hosted on the same target server. Any bean references obtained henceforth are also created on the same target host. This effectively provides load balancing, since all clients randomize the list of live target servers when creating `InitialContext` objects.

If the target server instance goes down, the lookup or EJB method invocation will fail over to another server instance. All objects derived from same `InitialContext` will fail over to the same server instance.

IIOP load balancing and failover happens transparently. No special steps are needed during application deployment. IIOP load balancing and failover for the Payara Server supports dynamically reconfigured clusters.

If the Payara Server instance on which the application client is deployed participates in a cluster, the Payara Server finds all currently active IIOP endpoints in the cluster automatically. Therefore, you are not required to manually update the list of endpoints if a new instance is added to the cluster or deleted from the cluster.

However, a client should have at least two endpoints specified for bootstrapping purposes, in case one of the endpoints has failed.

`InitialContext` Algorithm

Payara Server uses a randomization and round-robin algorithm for RMI-IIOP load balancing and failover.

When an RMI-IIOP client first creates a new `InitialContext` object, the list of available Payara Server IIOP endpoints is randomized for that client.

For that `InitialContext` object, the load balancer directs lookup requests and other `InitialContext` operations to an endpoint on the randomized list. If that endpoint is not available then a different random endpoint in the list is used.

Each time the client subsequently creates a new `InitialContext` object, the endpoint list is rotated so that a different IIOP endpoint is used for `InitialContext` operations. The rotation is randomized, so the rotation is not to the next endpoint in the list, but instead to a random endpoint in the list.

When you obtain or create beans from references obtained by an `InitialContext` object, those beans are created on the Payara Server instance serving the IIOP endpoint assigned to the `InitialContext` object. The references to those beans contain the IIOP endpoint addresses of all Payara Server instances in the cluster.

The primary endpoint is the bean endpoint corresponding to the `InitialContext` endpoint used to look up or create the bean.

The other IIOP endpoints in the cluster are designated as alternate endpoints. If the bean's primary endpoint becomes unavailable, further requests on that bean fail over to one of the alternate endpoints.

You can configure RMI-IIOP load balancing and failover to work with applications running in the ACC.

Enabling RMI-IIOP Hardware Load Balancing and Failover

You can enable RMI-IIOP load balancing and failover for applications running in the application client container (ACC). Weighted round-robin load balancing is also supported.

To Enable RMI-IIOP Hardware Load Balancing for the Application Client Container

This procedure provides an overview of the steps necessary to enable RMI-IIOP load balancing and failover with the application client container (ACC). For additional information on the ACC, see "developing-clients-using-the-acc" in Payara Server Application Development section.

Before You Begin

The first five steps in this procedure are only necessary if you are enabling RMI-IIOP load balancing on a system other than the DAS.

This is common in production environment, but less common in a development environment.

For example, a developer who wants to experiment with a cluster and load balancing might create two instances on the same system on which the DAS is running. In such cases, these steps are unnecessary.

1. Go to the `install_dir/bin` directory.
2. Run the `package-appclient` utility.

   This utility produces an `appclient.jar` file. For more information on `package-appclient`, see package-appclient.

3. Copy the `appclient.jar` file to the machine where you want your client and extract it.
4. Edit the `asenv.conf` or `asenv.bat` path variables to refer to the correct directory values on that machine.

   The file is at `appclient-install-dir/config/`.

   For a list of the path variables to update, see "`package-appclient`".
5. If required, make the `appclient` script executable.

   For example, on UNIX use `chmod 700`.
6. Find the IIOP listener port number for at least two instances in the cluster.

   You specify the IIOP listeners as endpoints in add-at-least-two-target-server-elements-in-the-sun-acc.xml-file.

   For each instance, obtain the IIOP listener ports as follows:

   a. Verify that the instances for which you want to determine the IIOP listener port numbers are running.

   ```
   asadmin> list-instances
   ```

   A list of instances and their status (running, not running) is displayed. The instances for which you want to display the IIOP listener ports must be running.

   b. For each instance, enter the following command to list the various port numbers used by the instance.

   ```
   asadmin> get servers.server.instance-name.system-property.*.value
   ```

   For example, for an instance name `in1`, you would enter the following command:

   ```
   asadmin> get servers.server.in1.system-property.*.value
   ```

7. Add at least two `target-server` elements in the `sun-acc.xml` file.

   Use the endpoints that you obtained in find-the-iiop-listener-port-number-for-at-least-two-instances-in-the-cluster.

   If the Payara Server instance on which the application client is deployed participates in a cluster, the ACC finds all currently active IIOP endpoints in the cluster automatically. However, a client should have at least two endpoints specified for bootstrapping purposes, in case one of the endpoints has failed.

   The `target-server` element specifies one or more IIOP endpoints used for load balancing. The `address` attribute is an IPv4 address or hostname, and the `port` attribute specifies the port number. See "client-container" in the Payara Server Application Deployment section.

   As an alternative to using `target-server` elements, you can use the `endpoints` property as follows:

   ```
   jvmarg value = "-Dcom.sun.appserv.iiop.endpoints=host1:port1,host2:port2,..."
   ```

8. If you require weighted round-robin load balancing, perform the following steps:

   a. Set the load-balancing weight of each server instance.

      ```
      asadmin set instance-name.lb-weight=weight
      ```

   b. In the `sun-acc.xml`, set the `com.sun.appserv.iiop.loadbalancingpolicy` property of the ACC to `ic-based-weighted`.

      ```
      <client-container send-password="true">
        <property name="com.sun.appserv.iiop.loadbalancingpolicy" value="ic-based-weighed"/>
      </client-container>
      ```

9. Deploy your client application with the `--retrieve` option to get the client jar file. Keep the client jar file on the client machine. For example:

```
asadmin  --user admin --passwordfile pw.txt deploy --target cluster1 --retrieve
my_dir myapp.ear
```

10. Run the application client as follows:

```
appclient --client my_dir/myapp.jar
```

In the following example, the load-balancing weights in a cluster of three instances are to be set as shown in the following table.

Instance Name	Load-Balancing Weight
`i1`	100
`i2`	200
`i3`	300

The sequence of commands to set these load balancing weights is as follows:

```
asadmin set i1.lb-weight=100
asadmin set i2.lb-weight=200
asadmin set i3.lb-weight=300
```

Next Steps

To test failover, stop one instance in the cluster and see that the application functions normally. You can also have breakpoints (or thread sleep calls) in your client application.

To test load balancing, use multiple clients and see how the load gets distributed among all endpoints.

See Also

See enabling-the-high-availability-session-persistence-service for instructions on enabling the session availability service for a cluster or for a Web, EJB, or JMS container running in a cluster.

Per-Request Load Balancing (PRLB)

Per Request Load Balancing (PRLB) is a method for load balancing stateless EJBs that enables load-balancing for each request to an EJB instance. **PRLB** chooses the first node in a cluster to use on each request. By contrast, `InitialContext` (hardware) load balancing chooses the first node to use when the `InitialContext` is created, and each request thereafter uses the same node unless a failure occurred.

PRLB is enabled by means of the boolean `per-request-load-balancing` property in the `glassfish-ejb-jar.xml` deployment descriptor file for the EJB. If this property is not set, the original load balancing behavior is preserved.

IMPORTANT	PRLB is only supported for stateless session beans. Using PRLB with any other bean types will result in a deployment error.

Enabling Per-Request Load Balancing

You can enable Per-Request Load Balancing (PRLB) by setting the boolean `per-request-load-balancing` property to `true` in the `glassfish-ejb-jar.xml` deployment descriptor file for the EJB. On the client side, the `initContext.lookup` method is used to access the stateless EJB.

To Enable RMI-IIOP Per-Request Load Balancing for a Stateless EJB

This procedure describes how to enable PRLB for a stateless EJB that is deployed to clustered Payara Server instances. This procedure also provides a client-side example for accessing a stateless EJB that uses PRLB.

1. Choose or assemble the EJB that you want to deploy. In this example, an EJB named `TheGreeter` is used.

   For instructions on developing and assembling an EJB for deployment to Payara Server, refer to the following documentation:

   - "using-enterprise-javabeans-technology" in the Payara Server Application Development section

- "ejb-module-deployment-guidelines" in the Payara Server Application Deployment section
- "assembling-and-deploying-an-application-client-module" in the Payara Server Application Deployment section

2. Set the `per-request-load-balancing` property to `true` in the `glassfish-ejb-jar.xml` deployment descriptor file for the EJB.

   For more information about the `glassfish-ejb-jar.xml` deployment descriptor file, refer to "the-payara-ejb-jar.xml-file" in the Payara Server Application Deployment section.

   For example, the `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` file for a sample EJB named `TheGreeter` is listed below.

```
<payara-ejb-jar>
  <enterprise-beans>
    <unique-id>1</unique-id>
    <ejb>
        <ejb-name>TheGreeter</ejb-name>
        <jndi-name>greeter</jndi-name>
        <per-request-load-balancing>true</per-request-load-balancing>
    </ejb>
  </enterprise-beans>
</payara-ejb-jar>
```

3. Deploy the EJB.

   NOTE | If the EJB was previously deployed, it must be re-deployed.

   For instructions on deploying EJBs, refer to the following documentation: * "to-deploy-an-application-or-module" in the Payara Server Application Deployment section * "to-redeploy-an-application-or-module" in the Payara Server Application Deployment section

4. Verify the PRLB configuration by looking for the following `FINE` message in the CORBA log file:

```
Setting per-request-load-balancing policyfor EJB EJB-name
```

5. Configure a client application to access the PRLB-enabled EJB. For example:

```
public class EJBClient {

    public static void main(String[] args) {
        try {
            // only one lookup
            var reference = initContext.lookup
("test.cluster.loadbalancing.ejb.TestSessionBeanRemote");
            var myGreeterRemote = (TestSessionBeanRemote)PortableRemoteObject
.narrow(reference, TestSessionBeanRemote.class);

        } catch (Exception e) {}

        for (int i=0; i < 10; i++ ) {
            var message = myGreeterRemote.sayHello(Integer.toString(i));
            System.out.println("got"+": " + message);
        }
    }
}
```

See Also

See enabling-the-high-availability-session-persistence-service for instructions on enabling the session availability service for a cluster or for a Web, EJB, or JMS container running in a cluster.

Security Guide

Overview

This security guide provides instructions for configuring and administering basic security concerns in Payara Server.

See Also

- Jakarta EE Tutorial: Working with Realms, Users, Groups, and Roles for similar configurations that are done in Eclipse GlassFish.

Administering System Security

This chapter describes general information about administering system security.

Information on application security is contained in "securing-applications" in the

Application Development section.

About System Security in Payara Server

Security is about protecting data, that is, how to prevent unauthorized access or damage to data that is in storage or in transit. Payara Server is built on the Java security model, which uses a sandbox where applications can run safely, without potential risk to systems or users.

System security affects all the applications in a Payara Server environment.

The Jakarta Security specification defines portable, plug-in interfaces for authentication and identity stores, and a new injectable-type `SecurityContext` interface that provides an access point for programmatic security. You can use the built-in implementations of the plug-in SPIs, or write your own custom implementations.

Authentication

Authentication is the way in which an entity (a user, an application, or a component) determines that another entity is who it claims to be. An entity uses security credentials to authenticate itself. The credentials might be a username and password, a digital certificate, or something else.

Usually, servers or applications require clients to authenticate themselves. Additionally, clients might require servers to authenticate themselves. When authentication is bidirectional, it is called **mutual authentication**.

When an entity tries to access a protected resource, Payara Server uses the authentication mechanism configured for that resource to determine whether to grant access. For example, a user can enter a set of credentials in a web browser, and if the application verifies those credentials, the user is authenticated. The user is associated with this authenticated security identity for the remainder of the session.

Authentication Types

Within its deployment descriptors, an application can specify the type of authentication that it uses. The Jakarta Security API provides an alternative mechanism for configuring

the type of authentication an application uses.

Payara Server supports specifying the following types of authentication in deployment descriptors:

BASIC: Uses the standard `Basic` Authentication Scheme as described in RFC2617. The communication protocol is HTTP (secure transport is optional).

+ There is no encryption of user credentials unless using secure transport.

+ This type is not considered to be a secure method of user authentication unless used in conjunction with an encrypted communications channel, such as that provided by SSL/TLS.

FORM: The application provides its own custom login and error webpages. The communication protocol is HTTP (secure transport is optional).

+ There is no encryption of user credentials unless using secure transport.

CLIENT-CERT: The server authenticates the client using a public key certificate.The communication protocol is HTTPS (secure transport required).

DIGEST: The server authenticates a user based on a username and a password. Unlike `BASIC` authentication, the plaintext password is never sent over the network, although a hash of the password combined with other parameters is sent.

+ While `Digest` Authentication is more secure than `Basic` Authentication, usernames and passwords are not strongly protected, and the use of secure transport is heavily recommended.

Jakarta Security Authentication Mechanisms and Identity Stores

The Jakarta Security API defines the `HttpAuthenticationMechanism` interface, the `IdentityStore` and `IdentityStoreHandler` interfaces.

The `HttpAuthenticationMechanism` interface defines an SPI for writing authentication mechanisms that can be provided with an application and deployed

using CDI. Developers can write their own implementations of `HttpAuthenticationMechanism` to support specific authentication token types or protocols. There are also several built-in authentication mechanisms that perform `BASIC`, `FORM`, and Custom `FORM` authentication.

The `HttpAuthenticationMechanism` interface defines three methods - `validateRequest()`, `secureResponse()`, and `cleanSubject()`. These methods align closely with the methods defined by the Jakarta Authentication's `ServerAuth` interface.

The `IdentityStore` interface provides an abstraction of an identity store that holds user account information including name, password, group membership, and other additional attributes. Implementations of the `IdentityStore` interface are used to validate caller credentials, typically username and password, and retrieve and group information. There are built-in implementations of this interface that can validate credentials against external **LDAP** or **Database** identity stores.

`IdentityStore` is intended primarily for use by `HttpAuthenticationMechanism` implementations, but could be used by other authentication mechanisms, such as a Jakarta Authentication `ServerAuthModule` implementation, or a runtime built-in authentication mechanisms. Though `HttpAuthenticationMechanism` implementations can authenticate users in any manner they choose, the `IdentityStore` interface provides a convenient mechanism.

A significant advantage of using `HttpAuthenticationMechanism` and `IdentityStore` over the declarative mechanisms defined by the Jakarta Servlet specification is that it allows an application to control the identity stores that it authenticates against, in a standard, portable way. You can use the built-in implementations of the plug-in SPIs, or define custom implementations.

Passwords

Passwords are your first line of defense against unauthorized access to the components and data of Payara Server. For Information about how to use passwords

for Payara Server, see administering-passwords.

Master Password and Java Keystores

The master password is not tied to a user account, and it is not used for authentication. Instead, Payara Server strictly uses the master password to ONLY encrypt the keystore and truststore used to store keys and certificates for the DAS and instances usage.

When you create a new Payara Server domain, a new self-signed certificate is generated and stored in the domain keystore and truststore. The DAS needs the master password to open both stores at startup. Similarly, the associated server instances need the master password to open their copy of these stores at startup.

If you use a utility such as `keytool` to modify the keystore or truststore, you must provide the master password in that case as well.

NOTE	The default master password (unless changed at domain creation) is `changeit`.
IMPORTANT	The master password is a shared password and must be the same for the DAS and all instances in the domain in order to manage the instances from the DAS. However, because Payara Server never transmits the master password over the network, it is up to you to keep the master password in sync between the DAS and instances.

If you change the master password, you can choose to enter the master password manually when required, or save it in a file.

Understanding Master Password Synchronization

Payara Server keeps the keystore and truststore for the DAS and instances in sync, which guarantees that all copies of the stores are encrypted with the same master password at any given time.

However, Payara Server **does not synchronize** the master password itself, and it is

possible that the DAS and instances might attempt to use different master passwords.

Consider the following potential scenario:

1. You create a domain and instances, using the default master password. As a result, the DAS and instances have keystores and truststores encrypted using `changeit`.
2. You use the `change-master-password` subcommand on the DAS to change the master password to `ichangedit`. As a result, the DAS and instance keystores and truststores are encrypted using `ichangedit`.
3. Access to the keystore and truststore from an instance now requires the master password `ichangedit`. You are responsible for changing the master password in each corresponding instance as needed.

If you do not use a master password file, you assume the responsibility for using the `change-master-password` subcommand on the DAS and instances to keep the master passwords in sync.

If you do use a master password file, you assume the responsibility for using the `change-master-password` subcommand on the DAS and instances to keep the master password file in sync.

Using the Default Master Password

Payara Server uses the known text string `changeit` as the default master password. This master password is not stored in a file. The default password is a convenience feature and provides no additional security because it is assumed to be widely known.

IMPORTANT	It is recommended to change the default master password in most production environment settings to prevent unwanted access to the server's configured keystore and truststores.

All Payara Server subcommands work as expected with the default master password and there are no synchronization issues.

Saving the Master Password to a File

The `change-master-password --savemasterpassword` option indicates whether the master password should be written to the file system in the `master-password` file for the DAS or a node. The default setting is `false`.

- For a domain, the master password is kept in the `domain-dir/master-password` file.
- For a node, the master-password file is kept in `as_install/nodes/<node-name>/agent/master-password`.

  You can set a master password at the node level and all instances created on that node will use that master-password file. To do this, use the `--nodedir` option and provide the corresponding node name.

You might want to save the master password to the file so that the `start-domain` subcommand can start the server without having to prompt the user.

WARNING	The master-password file is encoded, not encrypted. You must use filesystem permissions to protect the file.

Using the Master Password When Creating a Domain

The `create-domain --usemasterpassword` option specifies whether the keystore and truststore are encrypted with a master password that is built into the system, or by a user-defined master password.

- If `false` (default), the keystore is encrypted with the default master password.
- If `true`, the subcommand obtains the master password from the `AS_ADMIN_MASTERPASSWORD` entry in the password file you specified in the `--passwordfile` option of the `asadmin` utility. Or, if none is defined, the `--usemasterpassword` option prompts the user for the master password.

Administration Password

An administration password, also known as the admin password, is used to invoke the

Administration Console and the Asadmin CLI utility. As with the default `admin` username, the default admin password is usually set during installation, but it can be changed.

For instructions, see to-change-an-administration-password.

Encoded Passwords

Files that contain encoded passwords need to be protected using file system permissions. These files include the following:

- `domain-dir/master-password`

  This file contains the encoded master password and should be protected with the default file system permissions set to `600` (in most *Nix systems).

- Any password file created to pass as an argument by using the `--passwordfile` argument to the `asadmin` CLI.

  Additionally, any password file being used for a transient purpose, such as setting up SSH nodes, should be deleted after it has served its purpose.

For instructions, see to-set-a-password-from-a-file.

Web Browsers and Password Storage

Most web browsers can save login credentials entered through HTML forms. This function can be configured by the user and also by applications that employ user credentials. If the function is enabled, credentials entered by the user are stored on their local computer and retrieved by the browser on future visits to the same application.

This function is convenient for users, but can also be a security risk. The stored credentials can be captured by an attacker who gains access to the computer, either locally or through some remote compromise. Further, methods have existed whereby a malicious website can retrieve the stored credentials for other applications, by exploiting browser vulnerabilities or through application-level cross-domain attacks.

It is heavily recommended to disable password storage when prompted for the password of the Administration Console's web interface.

Password Aliases

To avoid storing passwords in the domain configuration file in clear text, you can create an alias for a password. This process is also known as encrypting a password. For more information, see administering-password-aliases.

Authorization

Authorization, also known as access control, is the means by which users are granted permission to access data or perform operations. After a user is authenticated, the user's level of authorization determines what operations they can perform. A user's authorization is based on the user's roles.

Roles

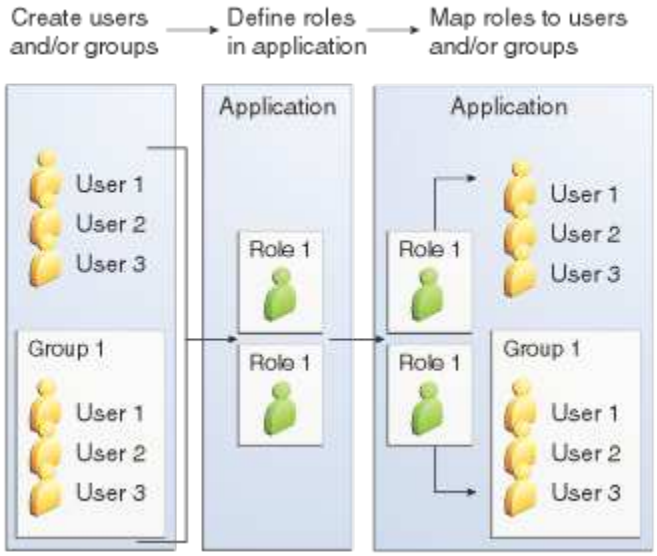
A role defines which applications and what parts of each application users can access and what those users or groups can do with the applications. For example, in a personnel application, all employees might be able to see phone numbers and email addresses, but only managers have access to salary information. This application would define at least two roles: `employee` and `manager`. Only users in the `manager` role are allowed to view salary information.

A role is different from a group in that a role defines a function in an application, while a group is a set of users who are related in some way.

For example, the personnel application specify groups such as`full-time`, `part-time`, and `on-leave`. Users in these groups are all employees (the `employee` role). In addition, each user has its own designation that defines an additional level of employment.

Roles are defined in the standard deployment descriptor for the application. The application developer or deployer maps roles to one or more groups in the deployment descriptor for each application. When the application is being packaged and deployed, the application specifies mappings between users, groups, and roles, like this:

Role Mappings



IMPORTANT

By default, group principal names are mapped to roles of the same name. Therefore, the Default Principal To Role Mapping setting is enabled by default on the *Security* page of the Administration Console.

With this setting enabled, if the group name defined on Payara Server matches the role name defined in the application, there is no need to use the runtime deployment descriptor to provide a mapping. The application server will implicitly make this mapping, as long as the names of the groups and roles match (case sensitivity considered).

Jakarta Authorization

Jakarta Authorization is the specification that defines an interface for pluggable policy providers. This enables you to set up third-party plug-in modules to perform authorization based on custom policies.

Working with the `server.policy` Policy File

Each Payara Server domain has its own global Java policy file, located in `domain-dir/config`. The file is named `server.policy`.

A sample `server.policy` file is as follows. Comments in the file describe why various permissions are granted. These permissions are described in more detail in the next section.

```
// classes in lib get all permissions by default
grant codeBase "file:${com.sun.aas.installRoot}/lib/-" {
    permission java.security.AllPermission;
};

// Core server classes get all permissions by default
grant codeBase "file:${com.sun.aas.installRoot}/modules/-" {
    permission java.security.AllPermission;
};

// Apache Felix classes get all permissions by default
grant codeBase "file:${com.sun.aas.installRoot}/osgi/felix/bin/-" {
    permission java.security.AllPermission;
};

// iMQ classes get all permissions by default
grant codeBase "file:${com.sun.aas.imqLib}/-" {
    permission java.security.AllPermission;
};

// H2 driver classes get all permissions by default
grant codeBase "file:${fish.payara.aas.h2Root}/bin/-" {
    permission java.lang.RuntimePermission "createClassLoader";
    permission java.io.FilePermission       "<<ALL FILES>>", "read,write";
};

// Basic set of required permissions granted to all remaining code
// The permission FilePermission "<<ALL FILES>>", "read,write"
// allows all applications to read and write any file in the filesystem.

// It should be changed based on real deployment needs. If you know your
// applications just need to read/write a few directories consider removing
// this permission and adding grants indicating those specific directories.
// against the codebase of your application(s).
grant {
    permission java.util.PropertyPermission "*", "read,write";

    permission java.lang.RuntimePermission  "loadLibrary.*";
    permission java.lang.RuntimePermission  "queuePrintJob";
    permission java.net.SocketPermission    "*", "connect";
    permission java.io.FilePermission       "<<ALL FILES>>", "read,write";

    permission java.io.FilePermission "${java.io.tmpdir}${/}-", "delete";

    permission java.util.PropertyPermission "*", "read";

    permission java.lang.RuntimePermission    "modifyThreadGroup";
    permission java.lang.RuntimePermission    "getClassLoader";
    permission java.lang.RuntimePermission    "setContextClassLoader";
    permission javax.management.MBeanPermission "[com.sun.messaging.jms.*:*]", "*";
};
```

```
// The following grant block is only required by Connectors. If Connectors are not
in use the recommendation is to remove this grant.
grant {
    permission javax.security.auth.PrivateCredentialPermission
"jakarta.resource.spi.security.PasswordCredential * \"*\"","read";
};

// The following grant block is only required for Reflection. If Reflection is not
in use the recommendation is to remove this section.
grant {
    permission java.lang.RuntimePermission "accessDeclaredMembers";
};

// Permissions to invoke CORBA objects in server
grant {
    permission com.sun.enterprise.security.CORBAObjectPermission "*", "*";
};

grant {
    permission java.io.SerializablePermission "enableSubclassImplementation";
};

grant {
    permission javax.management.MBeanPermission "sun.management.*", "*";
    permission javax.management.MBeanPermission "javax.management.*", "*";
    permission javax.management.MBeanPermission "[amx:*]", "*";
    permission javax.management.MBeanPermission "[amx-support:*]", "*";
    permission java.lang.management.ManagementPermission "monitor";
    permission javax.management.MBeanPermission "com.hazelcast.*", "*";
    permission javax.management.MBeanPermission "fish.payara.*", "*";
};

grant  codeBase "file:${com.sun.aas.instanceRoot}/applications/-"{
    permission java.io.FilePermission       "<<ALL FILES>>", "read,write";
};
```

Changing the Default Permissions

The Payara Server internal server code is granted all permissions. These grants are covered by the `AllPermission` grant blocks to various parts of the server infrastructure code. Do not modify these entries.

Application permissions are granted in the default `grant` block. These permissions apply to all code not part of the internal server code listed previously.

The section beginning with the comment *"Basic set of required permissions…"* provides the basic set of permissions granted to all remaining code.

Specifically, the following permission allows all applications to read and write all properties and read and write all files on the file system:

```
permission java.util.PropertyPermission "*", "read,write";
permission java.io.FilePermission       "<<ALL FILES>>", "read,write";
```

While this grant provides optimum flexibility, it is inherently unsecure. For enhanced security, change this permission based on your real deployment needs.

For example, consider removing this permission and assign default read and write permissions only to the application's install directory (context-root). This example uses `com.sun.aas.instanceRoot`, which specifies the top level directory for a server instance:

```
grant codeBase "file:${com.sun.aas.instanceRoot}/applications/MyApp/-"{
    permission java.io.FilePermission "file:${com.sun.aas.instanceRoot}
/applications/MyApp/-", "read,write";
}
```

For any application that needs to read and write additional directories, you would then have to explicitly allow such permissions by adding specific grants. In general, you should add extra permissions only to the applications or modules that require them, not to all applications deployed to a domain.

Firewalls

A firewall controls the flow of data between two or more networks, and manages communications between the networks. A firewall can consist of both hardware and software elements. The following guidelines pertain primarily to Payara Server:

- In general, firewalls should be configured so that clients can access the necessary TCP/IP ports.

  For example, if the HTTP listener is operating on port `8080`, configure the firewall to allow HTTP requests on port `8080` only. Likewise, if HTTPS requests are set up for port `8081`, you must configure the firewalls to allow HTTPS requests on port `8081`.

- If direct Remote Method Invocations over Internet Inter-ORB Protocol (RMI-IIOP) access from the Internet to EJB modules is required, open the ORB listener port as well.

CAUTION	Opening the ORB port to all possible clients is strongly discouraged because it creates security risks. Instead, create a valid access list of remote hosts that should access remote EJB modules and configure the firewall to grant them specific access.

- In a "double firewall" architecture, you must configure the outer firewall to allow for HTTP and HTTPS transactions. You must configure the inner firewall to allow the HTTP server plug-in to communicate with Payara Server behind the firewall.

Certificates and Secure Transport

Certificates

Certificates, also known as **digital certificates**, are electronic files that uniquely identify users and resources on the Internet and public networks. Certificates also enable secure, confidential communication between two entities. Usually, there are two different kinds of certificates:

- Personal certificates, used by individuals.
- Server certificates, used to establish secure-transport sessions between the server and clients through secure sockets layer (SSL) and transport layer security (TLS) technologies.

Certificates are based on **public key cryptography**, which uses pairs of digital keys to encrypt, or encode, information so the information can be read only by its intended recipient. The recipient then decrypts the information to read it.

A key pair contains a public key and a private key. The owner distributes the public key and makes it available to anyone, which is embedded on a digital certificate. The private key on the other hand is always kept secret. Because the keys are mathematically related, data encrypted with one key can only be decrypted with the other key in the pair.

Certificates are issued by a trusted third party called a **Certification Authority (CA)**. The CA is analogous to a passport office: it validates the certificate holder’s identity and signs the certificate so that it cannot be forged or tampered with. After a CA has signed

a certificate, the holder can present it as proof of identity and to establish encrypted, confidential communications.

Most importantly, a certificate binds the owner's public key to the owner's identity.

In addition to the public key, a digital certificate typically includes information such as the following:

- The name of the holder and other identification, such as the URL of the web server using the certificate, or an individual's email address
- The name of the CA that issued the certificate
- The certificate's expiration date

Certificates are governed by the technical specifications of the **X.509 format**. To verify the identity of a user in the `certificate` realm, the authentication service verifies an X.509 certificate, using the common name field (`CN`) of the X.509 certificate as the principal name.

Certificate Chains

A certificate chain is a series of certificates issued by successive CA certificates, eventually ending in a root CA certificate.

Most modern web browsers are preconfigured with a set of root CA certificates that the browser automatically trusts. Any certificates from elsewhere must come with a certificate chain to verify their validity.

When a certificate is first generated, it is a called a **self-signed certificate**. A self-signed certificate is one for which the issuer (signer) is the same as the subject (the entity whose public key is being authenticated by the certificate). When the owner sends a certificate signing request (CSR) to a CA, then imports the response, the self-signed certificate is replaced by a chain of certificates.

At the bottom of the chain is the certificate (reply) issued by the CA authenticating the subject's public key. The next certificate in the chain is one that authenticates the CA's

public key. Usually, this is a self-signed certificate (that is, a certificate from the CA authenticating its own public key) and the last certificate in the chain.

In other cases, the CA can return a chain of certificates. In this situation, the bottom certificate in the chain is the same (a certificate signed by the CA, authenticating the public key of the key entry), but the second certificate in the chain is a certificate signed by a different CA, authenticating the public key of the CA to which you sent the CSR. Then, the next certificate in the chain is a certificate authenticating the second CA's key, and so on, until a self-signed root certificate is reached.

Each certificate in the chain (after the first) thus authenticates the public key of the signer entity of the previous certificate in the chain.

Certificate Files

During Payara Server installation, a self-signed certificate is generated in Java Secure Socket Extension (JSSE) format suitable for internal testing. By default, Payara Server stores its certificate information in certificate databases in the `domain-dir/config` directory:

Keystore file: The `keystore.jks` file contains the certificate, including its private key. The keystore file is protected with a password.
Each keystore entry has a unique alias. After installation, the Payara Server keystore has a single entry with an alias of `s1as` that corresponds to the self-signed certificate and its key.

Truststore file: The `cacerts.p12` file contains trusted certificates, including public keys for other entities. For a trusted certificate, the server has confirmed that the public key in the certificate belongs to the certificate's owner. These trusted certificates generally include those of recognized CAs.

Secure Sockets Layer

Secure Sockets Layer (SSL) is a standard for securing Internet communications and transactions. Secure web applications use HTTPS (HTTP over SSL). The HTTPS protocol uses certificates to ensure confidential and secure communications between

server and clients. In an SSL connection, both the client and the server encrypt data before sending it. Data is decrypted upon receipt.

When a Web browser (client) wants to connect to a secure site, an SSL handshake happens, like this:

1. The browser sends a message over the network requesting a secure session (typically, by requesting a URL that begins with `https` instead of `http`).
2. The server responds by sending its certificate (including its public key).
3. The browser verifies that the server's certificate is valid and is signed by a CA whose certificate is in the browser's database (and who is trusted). It also verifies that the supplied certificate has not expired and is still valid.
4. If the certificate is valid, the browser generates a one-time, unique session key and encrypts it with the server's public key. The browser then sends the encrypted session key to the server so that they both have a copy.
5. The server decrypts the message using its private key and recovers the session key to fully establish the handshake.

After the handshake, the client has verified the identity of the website, and only the client and the Web server have a copy of the session key. From this point forward, the client and the server use the session key to encrypt all their communications with each other. Thus, their communications are ensured to be secure.

To use SSL, Payara Server must have a certificate for each external interface or IP address that accepts secure connections. The HTTPS service of most web servers will not run unless a certificate has been installed. For instructions on applying SSL to HTTP listeners, see "configure-an-http-listener-for-ssl" in the Payara Server General Administration section.

Ciphers

A cipher is a cryptographic algorithm used for encryption or decryption. SSL and TLS protocols support a variety of ciphers used to authenticate the server and client to each

other, transmit certificates, and establish session keys.

Some ciphers are stronger and more secure than others. Clients and servers can support different cipher suites. During a secure connection, the client and the server agree to use the strongest cipher that they both have enabled for communication, so it is usually sufficient to enable all ciphers.

Name-Based Virtual Hosts

Using name-based virtual hosts for a secure application can be problematic. This is a design limitation of the SSL protocol itself. The SSL handshake, where the client browser accepts the server certificate, must occur before the HTTP request is accessed. As a result, the request information containing the virtual host name cannot be determined prior to authentication, and it is therefore not possible to assign multiple certificates to a single IP address.

If all virtual hosts on a single IP address need to authenticate against the same certificate, the addition of multiple virtual hosts probably will not interfere with normal SSL operations on the server. Be aware, however, that most browsers will compare the server's domain name against the domain name (corresponding to the `CN` entry) listed in the certificate, if any (applicable primarily to official, CA-signed certificates). If the domain names do not match, most browsers will display a warning and fail the validation. In general, only address-based virtual hosts are commonly used with SSL in a production environment.

Understanding the `AppservCertificateLoginModule` Class

Payara Server provides the `AppservCertificateLoginModule` class to allow developers to implement a custom implementation of a login module that interacts with client certificates. This class provides some convenience methods for accessing the certificates, the application name and so forth, and for adding the group principals to the subject.

The convenience methods include the following:

`getAppName()`: Returns the name of the application to be authenticated. This may be

useful when a single `LoginModule` has to handle multiple applications that use certificates. **`getCerts()`**: Returns the certificate chain as an array of `java.security.cert.X509Certificate` certificates. **`getX500Principal()`**: Returns the distinguished (`DN`) principal from the first certificate in the chain. **`getSubject()`**: Returns the subject that is being authenticated. **`commitUserAuthentication(final String[] groups)`**: This method sets the authentication status to success if the groups parameter is non-null.

+ WARNING: This method is called after the authentication has succeeded. If the authentication failed, do not call this method.

NOTE	You do not have to extend the convenience base class, you can extend the JAAS LoginModule `javax.security.auth.spi.LoginModule` instead if you so choose.

Example `AppservCertificateLoginModule` Code

The following example shows a custom implementation of a certificate login module using the `AppservCertificateLoginModule` class.

Take note of the following points from the example:

- The `getX500Principal()` method returns the subject(subject distinguished name) value from the first certificate in the client certificate chain as an `X500Principal`.
- From that `X500Principal`, the `getName()` method then returns a string representation of the X.500 distinguished name using the format defined in RFC 2253 (LDAP v3).
- The example uses the `getAppName()` method to determine the application name. It also determines the organizational unit (`OU`) from the distinguished name.
- The example concatenates the application name with the value of `OU`, and uses it as the group name in the `commitUserAuthentication` method.

```
public class CustomCertificateLoginModule extends AppservCertificateLoginModule {

    @Override
    protected void authenticateUser() throws LoginException {
        // Get the distinguished name from the X500Principal.
        var st = new StringTokenizer(getX500Principal().getName(), "B \t\n\r\f,");
        while (st.hasMoreTokens()) {
            var next = st.nextToken();
            /*
             Set appname:OU as the group name.
             At this point, one has the application name and the DN of
             the certificate. A suitable login decision can be made here.
             */
            if (next.startsWith("OU=")) {
                commitUserAuthentication(new String[]{getAppName() + ":" + next
.substring(3)});
                return;
            }
        }
        throw new LoginException("No OU found in the DN of the supplied
certificate");
    }
}
```

Setting the JAAS Context

After you create your LoginModule, you must plug it in to a jaas-context, which you then specify as a parameter to the certificate realm's configuration in Payara Server.

To do this, perform the following steps:

1. Specify a new jaas-context for the Certificate realm in the `domain-dir/config/login.conf` file. Using the previous example configuration would look like this:

   ```
   certRealm {
       fish.payara.security.examples.CustomCertificateLoginModule required;
   };
   ```

2. Specify this jaas-context as a parameter to the `set` subcommand in the `configs.config.server-config.security-service.auth-realm.certificate.property.jaas-context=`<jaas-context-name> property. For example:

   ```
   asadmin set configs.config.server-config.security-service.auth-
   realm.certificate.property.jaas-context=certRealm configs.config.server-
   ```

```
config.security-service.auth-realm.certificate.property.jaas-context=certRealm

Command set executed successfully.
```

3. Optionally, get the value you just set to make sure that it is correct.

```
asadmin get configs.config.server-config.security-service.auth-
realm.certificate.property.jaas-context

configs.config.server-config.security-service.auth-
realm.certificate.property.jaas-context=certRealm

Command get executed successfully.
```

Administering Passwords

There are multiple ways to administer passwords. You can rely on administrators to keep passwords secret and change the passwords regularly. You can set up files for storing passwords so that `asadmin` CLI subcommands can access these files rather than having users type the commands.

You can also encrypt passwords by setting up aliases so that sensitive passwords are not clearly visible in the `domain.xml` configuration file.

To Change the Master Password

The master password gives access to the keystore and truststore used by the domain. This password is not tied to an operating system user. You should treat this overall shared password as sensitive data to secure. Payara Server never uses it for authentication and never transmits it over the network.

When a domain is created, you can choose to type its master password manually when required, or to obscure it in a password file. If there is no password file, you are prompted for the master password. If there is a password file, but you want to change access to require prompting, remove the file. The default master password is `changeit`.

IMPORTANT	When changing the master password, it has to be changed on all configured nodes as well as on the DAS. The master password on nodes is only stored once in the node, for all instances that are on

hosted on that node.

Use the `change-master-password` subcommand in local mode to modify the master password.

CAUTION

If you change the master password and are not using a master password file, the `start-instance` and `start-cluster` subcommands are not able to determine the master password. In this case, you must start those instances locally by using the `start-local-instance` command.

When the master password is saved, it is saved in the `domain-dir/config/master-password` file.

Follow these steps to correctly change the master password on an existing domain:

1. Stop the domain first.
2. Change the master password for the domain by using the "`change-master-password`" subcommand.
   You will be prompted for the old and new passwords based on whether you have been previously logged into the domain.
3. Start the domain again.

The following example changes the master password for the `custom-domain` domain:

```
asadmin> change-master-password custom-domain
```

If you have already logged into the domain using the "`login`" subcommand, you are prompted for the new master password:

```
Please enter the new master password>
Please enter the new master password again>
```

If you are not logged into the domain, you are prompted for both the old and the new

master passwords:

```
Please enter the master password>
Please enter the new master password>
Please enter the new master password again>
```

Information similar to the following is displayed:

```
Master password changed for domain44ps
```

Additional Considerations When Starting Instances through CLI Subcommands

If you change the master password for the DAS, the `start-domain` and `start-local-instance` subcommands allow you to provide it during domain or instance startup in one of three ways:

- Via the master-password file
- By entering it interactively
- Via the Asadmin CLI `--passwordfile` option

The `start-instance`, `start-cluster` and `start-deployment-group` subcommands require additional considerations. If you create a domain with a master password other than the default, an associated remote instance, cluster or deployment group must have access to the master password in order to properly start.

However, for security reasons Payara Server never transmits the master password or the master password file over the network.

Consider the following scenario:

1. Change the master password on the DAS and save it with the `--savemasterpassword` option.
2. Create an instance on another host using the subcommand `create-instance`. Payara Server then copies the keystore and truststore from the DAS to the remote instance, but it DOES not copy the master password file.

3. Try to start the instance using the `start-instance` subcommand. An error results preventing the instance to correctly start.

The `start-instance` command is looking for the master-password file in the node directory on the instance's host machine, and it is not there by default. Therefore, the subcommand fails.

You can use the `change-master-password` subcommand to make sure the correct password is used in this password file, as described in the following section.

NOTE

The `start-instance`, `start-cluster` and `start-deployment-group` subcommands do not include any other way for you to provide the password. If you change the master password and are not using a master password file, the `start-instance` and `start-cluster` subcommands will not be able to determine the master password. In this case, you must start the instances locally by using `start-local-instance` command.

Remotely Starting an Instance Using a Password File

Assume that you have changed the master password on the DAS and you want to make the same change for all instances.

The `start-instance`, `start-cluster` and `start-deployment-group` subcommands automatically use the master password file if it exists in the instance filesystem. You can use the `change-master-password` subcommand to make sure the password file exists and that the correct password is used.

Here's a concrete set of steps that showcase how to manage the master password from a domain's creation:

1. From the DAS, create a domain and set the master password.

```
asadmin create-domain --savemasterpassword true <domain-name>
```

2. Start the domain.

```
asadmin start-domain <domain-name>
```

3. Create a remote SSH node.

```
asadmin create-node-ssh --nodehost <host-name> --installdir /usr/local/payara
<node-name>
```

4. Create a new instance on the node.

```
asadmin create-instance --node <node-name> <instance-name>
```

5. Before you start the instance, on the node's host machine run the `change-master-password` with the `---savemasterpassword` option to create a file called `master-password` in the `agents` directory so that the instance can correctly start up.

   Locally on the remote node's host:

```
asadmin change-master-password --savemasterpassword true --nodedir
/usr/local/payara <node-name>

Enter the current master password>
Enter the new master password>
Enter the new master password again>

Command change-master-password executed successfully.
```

   Remember that when you created the domain you specified a new master password. This master password was then used to encrypt the keystore and truststore for the DAS, and these stores were copied to the instance as a result of the `create-instance` subcommand.

   Therefore, enter the master password you set when you created the domain as both the current master password and again as the new master password. You enter it as the new master password because you do not want to change the master password for the instance and make it out of sync with the DAS.

6. Run the `start-instance` from the DAS.

```
asadmin start-instance <instance-name>
```

NOTE

The master password file is associated with the node and not with an instance. After the master password file exists in the node directory on the instance machine, additional instances can be created, started and stopped from the DAS with no additional considerations.

To Change an Administration Password

Use the `change-admin-password` subcommand in remote mode to change an administration password. The default administration user is `admin`. When run, you will be prompted for the old and new admin passwords, with confirmation.

When using the default domains shipped with Payara Server, the default administration user will have a blank password.

NOTE

If there is a single user called `admin` that does not have a password, you are not prompted for login information when running any Asadmin CLI commands that affect a running domain or access the Admin console.

WARNING

If secure administration is enabled as described in link:Technical Documentation/Payara Server Documentation/Security Guide/administrative-security.adoc#running-secure-admin[Running Secure Admin], you cannot change an administration password to a blank value under any circumstance.

1. Change the admin password by using the change-admin-password subcommand.
2. Enter the old and new admin passwords when prompted.
3. Restart the Payara Server domain.

This example changes the admin password for user anonymous from `adminadmin` to `newadmin`:

```
asadmin change-admin-password --username anonymous
```

You are prompted to enter the old and the new admin passwords:

```
Enter admin password>adminadmin
Enter new admin password>newadmin
Enter new admin password again>newadmin

Command change-admin-password executed successfully.
```

To Set a Password from a File

Instead of typing a specific password (like the admin password or the master password) at the command line, you can supply the password for a command from a text file such as `passwords.txt`.

The `--passwordfile` option of the Asadmin CLI utility takes the name of the file that contains the passwords. The entry for a specific password in the file must have the `AS_ADMIN_` prefix followed by the password name in uppercase letters.

NOTE

Any password file created to pass as an argument by using the `--passwordfile` argument to the `asadmin` utility should be protected with the corresponding file system permissions. Additionally, any password file being used for a transient purpose, such as setting up SSH among nodes, should be deleted after it has served its purpose.

For a complete list of the types of passwords that can be specified, see the "`asadmin`" command help page. Here are a couple of examples of password names used by the file.

```
AS_ADMIN_MASTERPASSWORD
AS_ADMIN_USERPASSWORD
AS_ADMIN_ALIASPASSWORD
```

For example, to specify an administrative password, add an entry similar to the following to the password file, where `adminadmin` is the current password:

```
AS_ADMIN_PASSWORD=adminadmin
```

Save your changes. You can now specify the password file in an `asadmin` subcommand. In this example, `passwords.txt` is the file that contains the password:

+

```
asadmin delete-jdbc-resource --user admin --passwordfile passwords.txt
jdbc/CustomPool
```

Administering Password Aliases

A password alias is used to indirectly access a password so that the password itself does not appear in cleartext in the domain's `domain.xml` configuration file.

Storing and passing passwords in cleartext is a security risk, and may violate some corporate security policies. In such cases, you can use password aliases to protect your passwords.

Create a Password Alias

Use the `create-password-alias` subcommand in remote mode to create an alias for a password in the domain. The password corresponding to the alias name is stored in an encrypted form in the domain configuration file.

The `create-password-alias` subcommand takes both a secure interactive form, in which users are prompted for all information, and a more script-friendly form, in which the password is propagated on the command line.

You can also use the "`set`" subcommand to remove and replace the password in the configuration file. For example:

```
asadmin set --user admin server.jms-service.jms-host.default_JMS_host.
admin-password='${ALIAS=jms-password}'
```

To create a new password alias follow these steps:

1. Run the "`create-password-alias`" subcommand.
2. Type the password for the alias when prompted.
3. Add the alias to a password file.

   For example, assume the use of a password file such as `passwords.txt`. Assume further that you want to add an alias for the `AS_ADMIN_USERPASSWORD` entry that is read by the "`create-file-user`" subcommand.

   You would add the following line to the password file:

   ```
   AS_ADMIN_USERPASSWORD=${ALIAS=user-password-alias}
   ```

   Where `user-password-alias` is the new password alias.

4. Now execute the corresponding command that requires the alias in the password file. Here's an example that uses the "`create-file-user`" subcommand:

   ```
   asadmin --passwordfile passwords.txt create-file-user user1
   ```

Here's another example that creates the new `jms-password` alias for the `admin` user:

```
asadmin> create-password-alias --user admin jms-password
```

Then, you are prompted to type the password for the alias:

```
Please enter the alias password> secret-password
Please enter the alias password again> secret-password
Command create-password-alias executed successfully.
```

Here's another example that creates a new password alias using the provided name. The user is then prompted to enter the associated password twice.

```
asadmin> create-password-alias example-alias-name
Enter the alias password>
Enter the alias password again>
Command create-password-alias executed successfully.
```

To List Password Aliases

Use the `list-password-aliases` subcommand in remote mode to list existing the password aliases.

This example lists all existing password aliases:

```
asadmin list-password aliases
jmspassword-alias

Command list-password-aliases executed successfully
```

To Delete a Password Alias

Use the `delete-password-alias` subcommand in remote mode to delete an existing password alias.

This example deletes the password alias `jmspassword-alias`:

```
asadmin> delete-password-alias jmspassword-alias

Command list-password-aliases executed successfully
```

To Update a Password Alias

Use the `update-password-alias` subcommand in remote mode to change the password for an existing password alias. The `update-password-alias` subcommand takes both a secure interactive form, in which the user is prompted for all information, and a more script-friendly form, and the password is propagated on the command line.

This example updates the password for the `jmspassword-alias` alias:

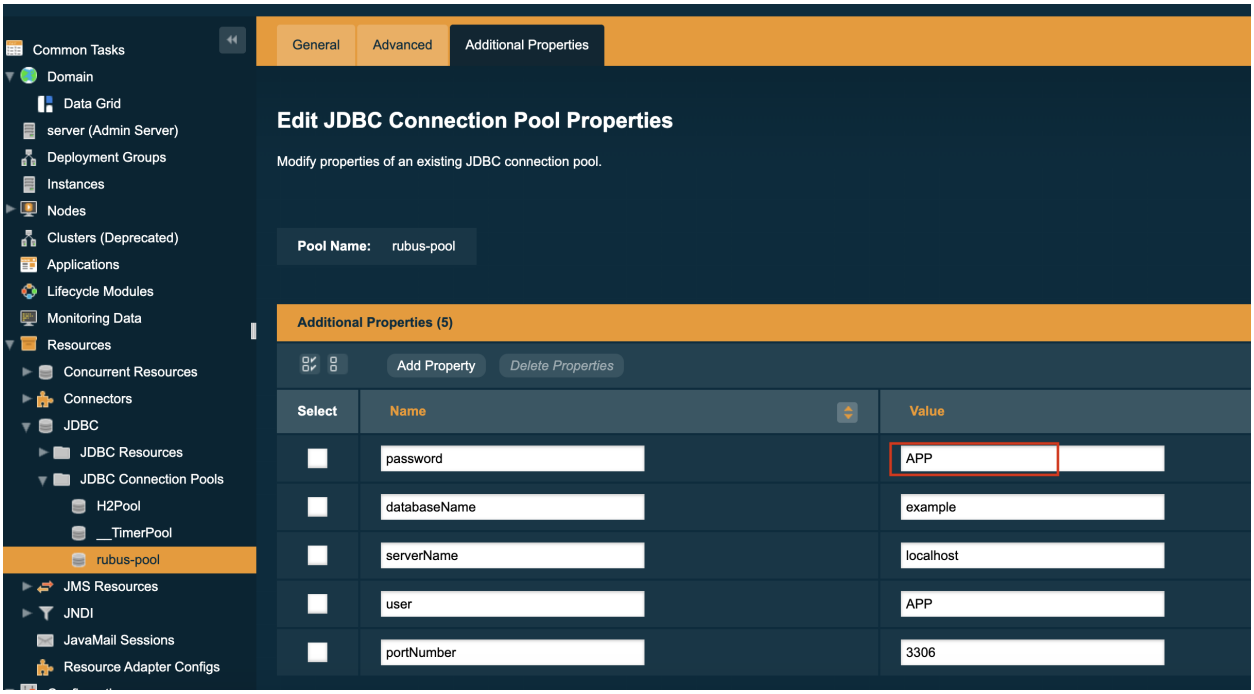
```
asadmin> update-password-alias jsmpassword-alias
```

You are prompted to type the new password for the alias:

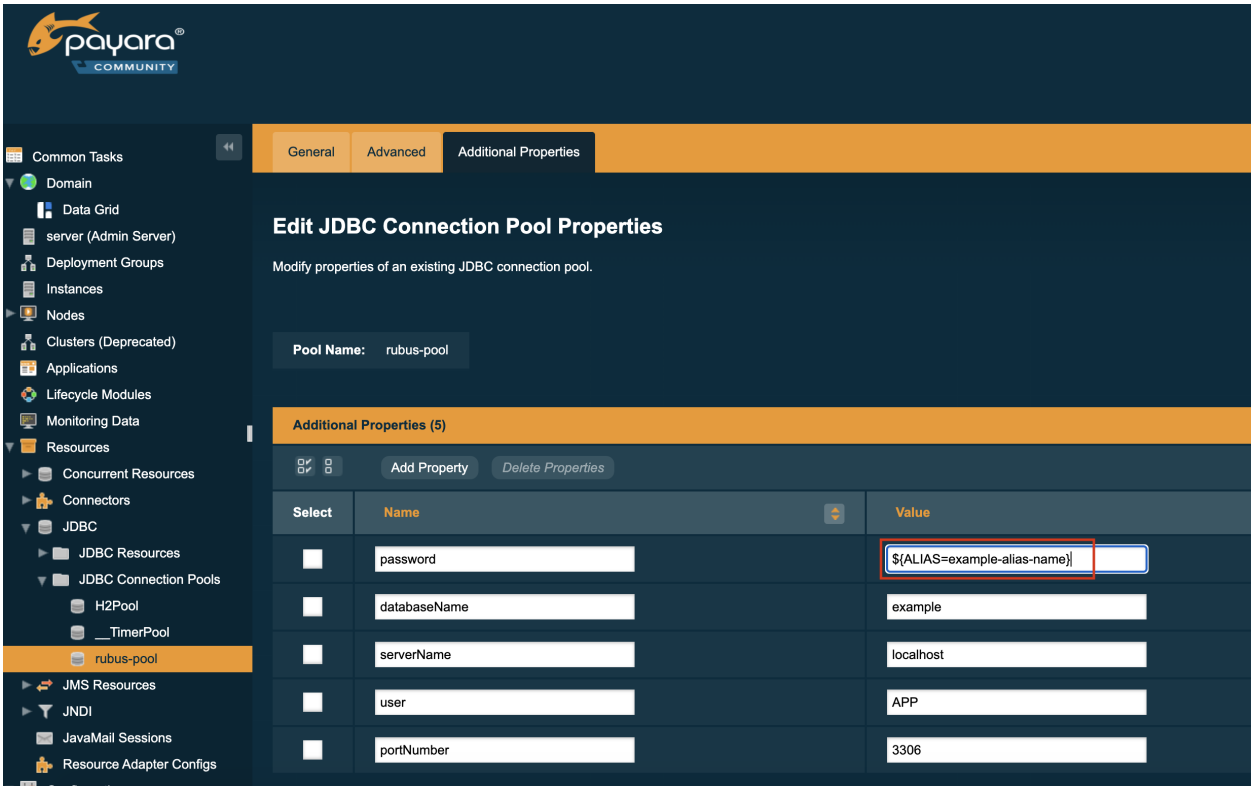
```
Please enter the alias password> new-secret-password
Please enter the alias password again> new-secret-password
Command update-password-alias executed successfully
```

Using a Password Alias within the Admin Console

To use a password alias, navigate to any configuration view where you would have previously entered a password in plaintext format:



In place of the password, you can enter an alias which corresponds to the password in the form: `${ALIAS=<password-alias-name>}`:

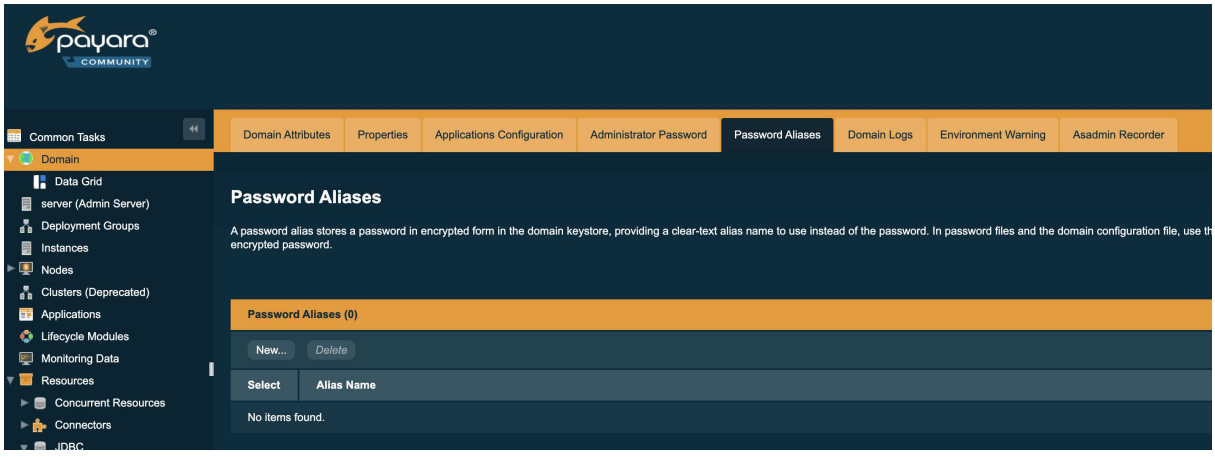


Managing Password Aliases through the Admin Console

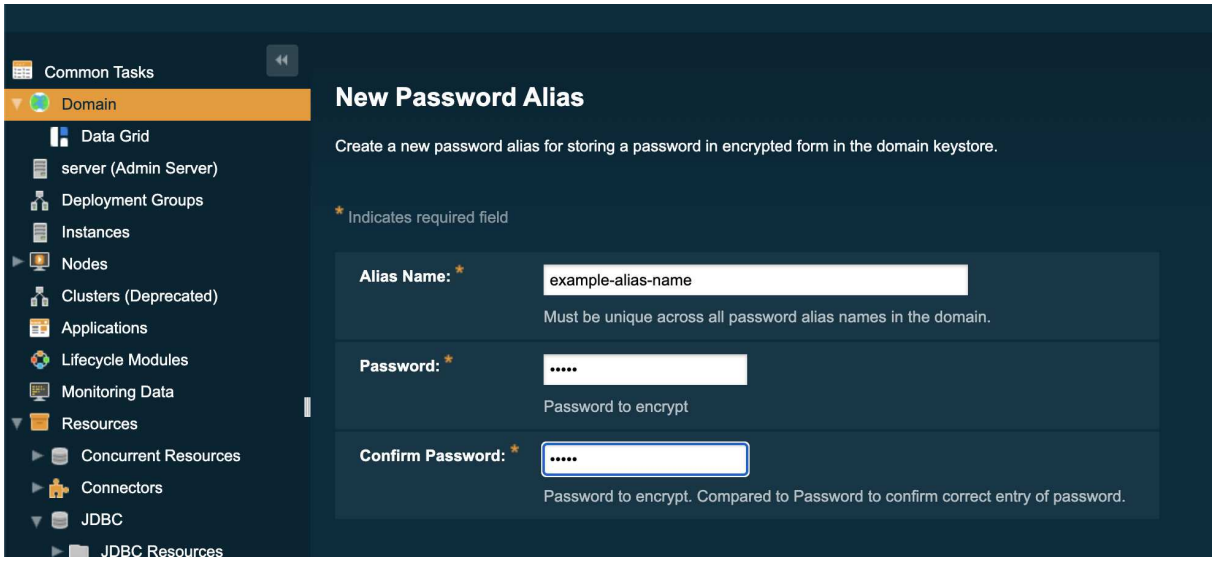
Here are instructions on managing password aliases through the web administration console.

Creating a Password Alias

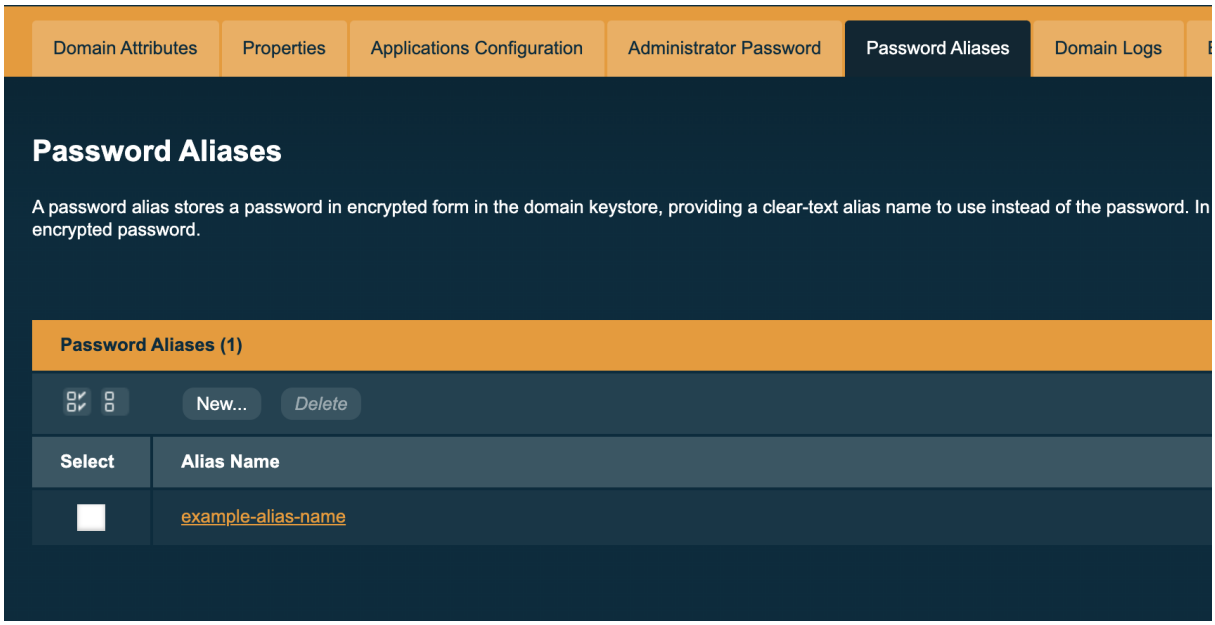
1. To create a new password alias for the domain, select **Domain** on the page tree, then select the **Password Aliases** tab and click the **New** button:



2. On the next page, enter the name of the alias, and the password, twice:

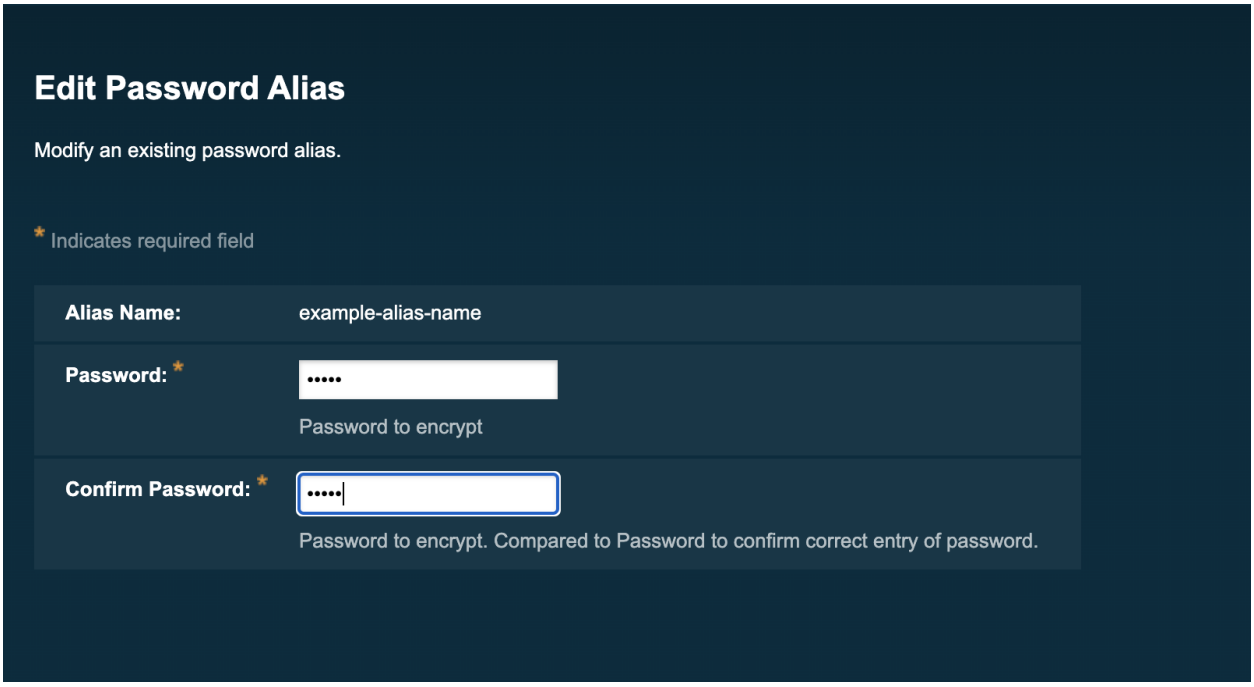


3. Finally, press the **OK** button to create your new alias:



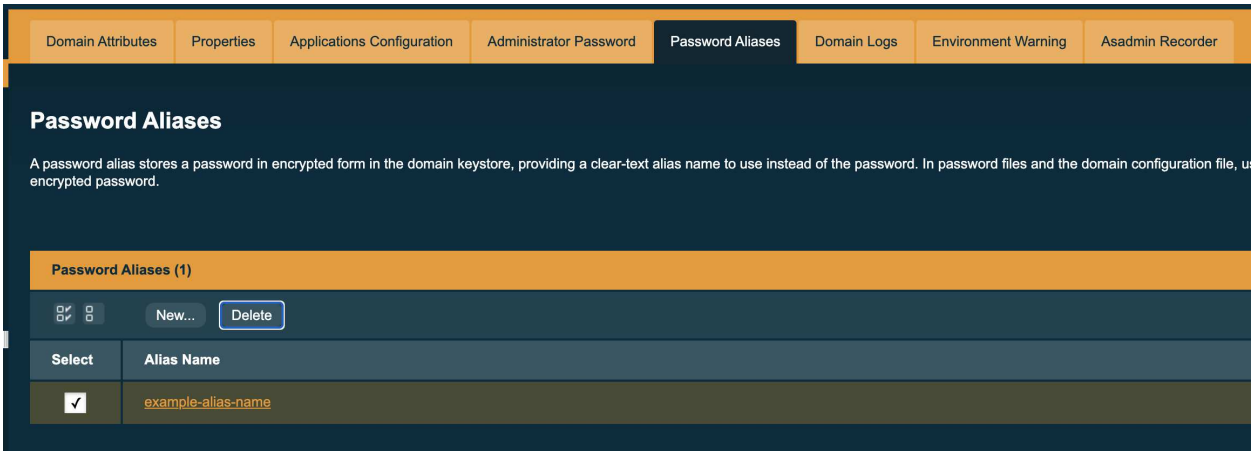
Modifying a Password Alias

To modify an existing password alias, select it on the **Password Aliases** page. On the **Edit Password Alias** page you are able to change the entered password.



Deleting a Password Alias

To delete a password alias, go to the **Password Aliases page**, tick the checkbox of the password alias you wish to delete and then press the **Delete** button:



Microprofile Config Support

Password aliases can also be accessed using MicroProfile Config, as detailed "here".

Administering JSSE Certificates

The Java SE SDK ships with the `keytool` utility, which enables you to set up and work with Java Secure Socket Extension (JSSE) digital certificates and similar formats. You can administer public/private key pairs and associated certificates, and cache the

public keys (in the form of certificates) of their communicating peers.

To Generate a Certificate by Using the `keytool` Utility

By default, the `keytool` utility creates a keystore file in the directory where the utility is run. The following are steps to generate a certificate using the `keytool` utility an import it in the domain's keystore and truststore. These commands should be run from the domain's configuration directory.

1. Change to the directory that contains the keystore and truststore files.

TIP	Always generate the certificate in the directory containing the keystore and truststore files.

2. Generate the certificate in the keystore file, `keystore.p12`, using the following command format:

   ```
   keytool -genkey -alias keyAlias -keyalg RSA -keypass changeit -storepass
   changeit -keystore keystore.p12
   ```

   Use any unique name as the alias. If you have changed the keystore or private key password from the default (`changeit`), substitute the new password for `changeit`. The default key password alias is `s1as`.

   A prompt will ask for your name, organization, and other information.

3. Export the generated certificate to the `server.cer` file, using the following command format:

   ```
   keytool -export -alias keyAlias -storepass changeit -file server.cer -keystore
   keystore.p12
   ```

4. If a certificate signed by a certificate authority is required, the following section.

5. Import the certificate to the truststore, using the following command:

   ```
   keytool -import -v -trustcacerts -alias keyAlias -file server.cer -keystore
   cacerts.p12 -storepass changeit -keypass changeit
   ```

If you have changed the keystore or private key password from the default (`changeit`), substitute the new password.

Information about the certificate is displayed and a prompt appears asking if you want to trust the certificate.

6. Type `yes`, then press Enter.

   Information similar to the following is displayed:

```
Certificate was added to keystore
[Saving cacerts.p12]
```

7. To apply your changes, restart the Payara Server domain and corresponding instances.

The following is an example on how to create a Self-Signed Certificate in a PKCS12 Keystore by Using an **RSA Key Algorithm**:

```
keytool -genkey -noprompt -trustcacerts -keyalg RSA -alias ${cert.alias} -dname
${dn.name} -keypass ${key.pass} -keystore ${keystore.file} -storepass ${
keystore.pass}
```

The following is an example on how to create a Self-Signed Certificate in a PKCS12 Keystore by using the default key algorithm:

```
keytool -genkey -noprompt -trustcacerts -alias ${cert.alias} -dname ${dn.name}
-keypass ${key.pass} -keystore ${keystore.file} -storepass ${keystore.pass}
```

The following command will display detailed information about all certificates present in a keystore:

```
keytool -list -v -keystore ${keystore.file} -storepass ${keystore.pass}
```

The following command will display detailed information about a specific certificate identified by the supplied alias in a keystore:

```
keytool -list -v -alias ${cert.alias} -keystore ${keystore.file}
-storepass ${keystore.pass}
```

Generate and Sign Certificate by Using `keytool`

After creating a certificate, the owner must sign the certificate to prevent forgery. E-commerce sites, or those for which authentication of identity is important, can purchase a signed certificate from a trustworthy Certificate Authority (CA) or third parties associated with one.

TIP If integrity is not a concern (in some rare cases), for example if private secure communications are all that is required, you can save the time and expense involved in obtaining a CA certificate by using a self-signed certificate.

The following steps outline how to create and sign a new certificate:

1. First, delete the default self-signed certificate:

   ```
   keytool -delete -alias s1as -keystore keystore.p12 -storepass <store_passwd>
   ```

   NOTE `s1as` is the default alias of the Payara Server keystore.

2. Generate a new key pair to be used by the certificate:

   ```
   keytool -genkeypair -keyalg <key_alg> -keystore keystore.p12 -validity
   <val_days> -alias s1as
   ```

   Where `<key_alg>` is the algorithm to be used for generating the key pair, for example `RSA`, and `<val_days>` is the number of days that the certificate should be considered valid.

   In addition to generating a key pair, the command wraps the public key into a self-signed certificate and stores the certificate and the private key in a new keystore entry identified by the inputted alias.

IMPORTANT	For HTTPS hostname verification, it is important to ensure that the name of the certificate (CN) matches the fully-qualified hostname of your site (fully-qualified domain name). If the names do not match, clients connecting to the server will see a security alert stating that the name of the certificate does not match the name of the site.

3. Generate a Certificate Signing Request (CSR):

```
keytool -certreq -alias s1as -file <certreq_file> -keystore keystore.p12
-storepass <store_passwd>
```

   Where `<certreq_file>` is the file in which the CSR is stored (for example, `s1as.csr`) and <store_passwd> is the password for the keystore (which corresponds to the domain's master password).

4. Submit the CSR to a Certificate Authority to get a signed certificate. In response, you should receive a signed server certificate. Make sure to import into your browser the CA certificate of the CA (if not already present) and any intermediate certificates indicated by the CA in the reply.
5. Download the CA certificate and any intermediate CA certificates and store them in local files.
6. Import the CA certificate (if not already present) and any intermediate CA certificates (if not already present) indicated by the CA into the domain's truststore:

```
keytool -import -v -trustcacerts -alias <CA-Name> -file ca.cert -keystore
cacerts.p12 -storepass <store_passwd>
```

7. Replace the original self-signed certificate with the certificate you obtained from the CA, as stored in a file such as `s1as.cert`:

```
keytool -import -v -trustcacerts -alias s1as -file s1as.cert -keystore
keystore.jks -storepass <store_passwd>
```

When you import the certificate using the same original alias `s1as`, the `keytool` utility treats it as a command to replace the original certificate with the certificate obtained as a reply to a CSR.

After running the command, you should see that the certificate `s1as` in the keystore is no longer the original self-signed certificate, but is now the certificate delivered by the CA.

Consider the following example that compares an original `s1as` certificate with a new `s1as` certificate obtained from VeriSign:

```
Original s1as (self-signed):

Owner: CN=demo.payara.fish, OU=Payara, O=Payara Foundation, L=Great Malvern,
ST=Worcestershire, C=UK
Issuer: CN=demo.payara.fish, OU=Payara, O=Payara Foundation, L=Great Malvern,
ST=Worcestershire, C=UK
Serial number: 472acd34
Valid from: Fri Nov 02 12:39:40 GMT+05:30 2023 until: Mon Oct 30 12:39:40
GMT+05:30 2033

New s1as (contains signed cert from CA):

Owner: CN=demo.payara.fish, OU=Payara, O=Payara Foundation, L=Great Malvern,
ST=Worcestershire, C=UK
Issuer: CN=VeriSign Trial Secure Server Test CA, OU=Terms of use at
https://www.verisign.com/cps/testca (c)05, OU="For Test Purposes Only. No
assurances.", O="VeriSign, Inc.", C=US
Serial number: 1375de18b223508c2cb0123059d5c440
Valid from: Sun Nov 11 05:30:00 GMT+05:30 2023 until: Mon Nov 26 05:29:59
GMT+05:30 2025
```

8. To apply your changes, restart the Payara Server domain.

To Delete a Certificate by Using `keytool`

Example 15 Importing an RFC/Text-Formatted Certificate Into a JKS Keystore

Use the `keytool delete` command to delete an existing certificate.

Here's an example on how to delete an existing certificate from the truststore using the following command:

```
keytool -delete -alias keyAlias -keystore cacerts.p12 -storepass password
```

Administering Jakarta Authorization / JACC Providers

The Jakarta Authorization specification (formerly known as Java Authorization Contract for Containers - JACC) defines an interface for pluggable authorization providers (commonly known as JACC providers) based on policies. This enables the administrator to set up third-party plug-in modules to perform authorization.

Payara Server includes Administration Console support and subcommands to support JACC providers, as follows:

- create `create-jacc-provider`
- delete `delete-jacc-provider`
- list `list-jacc-providers`

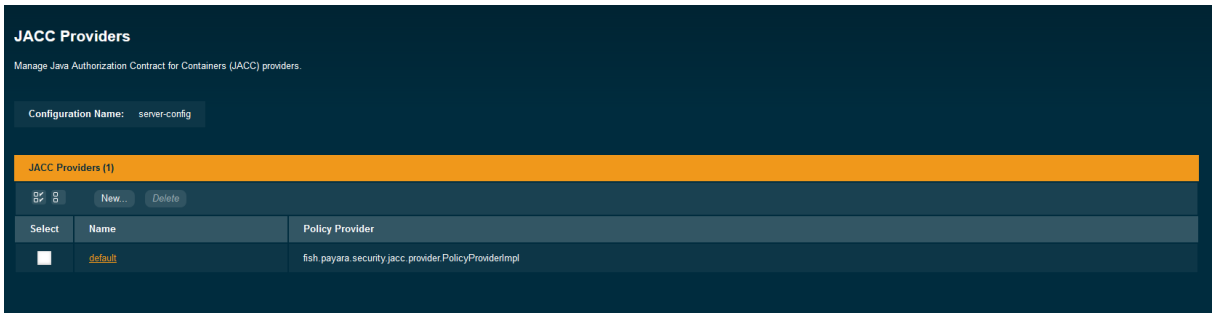
IMPORTANT	Payara Server includes a default JACC provider named `default` which you should not delete under any circumstance.

Payara Server allows Jakarta Authorization compliant JACC providers to be registered, so they can be used as third-party authorization modules for applications deployed on its corresponding domain.

Administering JACC Providers from the Administration Console

To use the Administration Console to administer JACC providers, perform the following steps:

1. Select *Configurations* and expand the entry.
2. Select the server configuration for which you want to administer JACC providers and expand the entry.
3. Select *Security* and expand the entry.
4. Select *JACC Providers*. The existing JACC providers are shown on this page.



5. To create a new provider, click *New*.

   Enter the **Name**, **Policy Configuration** (the class that implements the policy configuration factory) and the **Policy Provider** (the class that implements the policy factory) for the new JACC provider.

   You can also enter optional properties to further configure the provider.

6. To delete an existing JACC provider, select that provider and click Delete.

Administering JACC Providers from the Command Line

To use the Asadmin CLI to administer JACC providers, perform the following steps:

1. To create a JACC provider, use the `create-jacc-provider` subcommand. The following example shows how to create a JACC provider named `testJACC` on the default server target.

   ```
   asadmin create-jacc-provider --policyproviderclass
   fish.payara.security.jacc.sample.TestPolicyProvider --policyconfigfactoryclass
   fish.payara.security.jacc.provider.PolicyConfigurationFactoryImpl testJACC
   ```

2. To delete a JACC provider, use the `delete-jacc-provider` subcommand. The following example shows how to delete a JACC provider named `testJACC` from the default domain:

   ```
   asadmin delete-jacc-provider testJACC
   ```

3. To list the available providers, use the `list-jacc-providers` subcommand. The following example shows how to list JACC providers for a running domain:

```
asadmin> list-jacc-providers
default

Command list-jacc-providers executed successfully.
```

How to Install a Custom JACC Provider per Server (Global)

Having coded a JACC provider, the first thing to do it is to make these classes available for the server. For that purpose, you need to put the implementation JAR, with all its dependencies, under the `as-install/lib` folder.

The next thing to do is to tell Payara you want to use the custom JACC provider. To do this, you have to execute the following administration command:

```
asadmin create-jacc-provider --policyConfigurationFactoryProvider
=com.example.CustomPolicyConfigurationFactory --policyProvider
=com.example.CustomPolicy --target=server-config custom-provider set
configs.config.server-config.security-service.jacc=custom-provider
```

This will result in the following configuration element added to the *domain.xml* file:

```
<security-service jacc="custom-provider">
    <jacc-provider policy-provider="com.example.CustomPolicy" name="custom-
provider"
      policy-configuration-factory-
provider="com.example.CustomPolicyConfigurationFactory">
    </jacc-provider>
    <!-- More providers can be defined -->
</security-service>
```

As you can see on the XML excerpt, more JACC providers can be defined (by default, the `simple` and `default` providers are already defined), but only one will be used at runtime, specified by the `jacc` attribute on the `security-service` element.

How to Install a Custom JACC Provider per Application (Local)

With a custom JACC provide per server, applications deployed to a single server can't have their own custom authorization contracts. In this case, having a custom JACC Provider per application makes more sense by using the `fish.payara.jacc.JaccConfigurationFactory` API.

IMPORTANT	To access this API, you will need to add a dependency to the

> "Payara Public API" first.

In the example below, when an application is deployed, a custom JACC provider is registered programmatically using the `JaccConfigurationFactory (consisting of a JACC `PolicyConfigurationFactory` and `Policy` implementation objects) in the `contextInitialized` method of `ServletContextListener`.

```
@WebListener
public class JaccInstaller implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {

        JaccConfigurationFactory
                .getJaccConfigurationFactory()
                .registerContextProvider(
                        getAppContextId(sce.getServletContext()),
                        new CustomPolicyConfigurationFactory(),
                        new CustomPolicy()
                );

    }

    private String getAppContextId(ServletContext servletContext) {
        return servletContext.getVirtualServerName() + " " + servletContext
.getContextPath();
    }
}
```

Administering User Security

This chapter provides instructions for administering user security in the Payara Server environment by using the `asadmin` command-line utility. Payara Server enforces its authentication and authorization policies upon realms, users, and groups. This chapter assumes that you are familiar with security features such as authentication, authorization, and certificates. If you are not, see administering-system-security.

Instructions for accomplishing these tasks by using the Administration Console are contained in the Administration Console online help.

NOTE	Jakarta Security defines the concept of an Identity Store, and an SPI interface for writing providers that can authenticate users against Identity Stores. It also provides multiple built-in providers

> See Working with Identity Stores in The Jakarta EE Tutorial for more information about Identity Stores.

Administering Security Realms

Overview of Security Realms

A **security authentication realm**, also called a **security policy domain** or **security domain**, is a scope over which the Payara Server defines and enforces a common security policy. Payara Server is preconfigured with the file, certificate, and administration realms. In addition, you can set up LDAP, JDBC, digest, or custom realms. An application can specify which security realm to use in its deployment descriptors.

If the application does not specify a security realm, Payara Server uses its default realm (`file`).

File realm: Payara Server stores user credentials locally in a file named `keyfile`. This is the default realm.

Administration realm: The administration realm is also a file realm and stores administrator user credentials locally in a file named `admin-keyfile`.

Certificate realm: Payara Server stores user credentials in a certificate keystore. When using the certificate realm, the server uses certificates with the HTTPS protocol to authenticate web clients exclusively.

LDAP realm: Payara Server can get user credentials from a Lightweight Directory Access Protocol (LDAP) server.

JDBC realm: Payara Server gets user credentials from a database. The server uses the database information and the enabled JDBC realm option in the configuration file.

Digest realm: Digest Authentication authenticates a user based on a username and a password. However, the authentication is performed by transmitting the password in an encrypted manner using a digest algorithm.

Oracle Solaris realm: Payara Server gets user credentials directly from the Oracle

Solaris operating system user accounts.

+ IMPORTANT: This realm supports both Oracle Solaris 9 and 10 versions only.

PAM realm: A Pluggable Authentication Module (PAM) realm allows applications deployed on Payara Server to authenticate users against a native Unix users list. PAM realms are implemented through the `com.sun.enterprise.security.auth.realm.pam.PamRealm` class and the JAAS Context `pamRealm`.

Custom realm: You can create custom realms that retrieve user credentials from other type of repositories for user credentials, such as a NoSQL database or a third-party user management system. For more information about custom realms, see "creating-a-custom-realm" in the Payara Server Application Development section.

TIP	The Payara Server authentication service can govern users in multiple realms.

To Create an Authentication Realm

Use the `create-auth-realm` subcommand in remote mode to create an authentication realm.

1. Ensure that the server is running.
2. Create a realm by using the "`create-auth-realm`" subcommand.

The following example creates a realm named `custom-dbRealm` by using a customized realm implementation class named `fish.payara.security.auth.realm.DB.DatabaseRealm`:

```
asadmin create-auth-realm --classname
=fish.payara.security.auth.realm.DB.DatabaseRealm --property=group-name-column
=groupname:user-name-column=username:group-table=user_groups:user-table=
users:datasource-jndi=jdbc\/userDS:password-column=password:jaas-context=custom-db:
--login-module=fish.payara.support.SaltedSupportJDBCLoginModule custom-dbRealm

Command create-auth-realm executed successfully.
```

TIP	You can use the `--login-module` option to automatically register the JAAS login module in the server's policy file when the security realm is created.

Listing Authentication Realms

Use the `list-auth-realms` subcommand in remote mode to list existing authentication realms.

1. Ensure that the server is running.
2. List realms by using the "`list-auth-realms`" subcommand.

The following example lists the authentication realms registered on the DAS:

```
asadmin list-auth-realms

db
certificate
file
admin-realm

Command list-auth-realms executed successfully.
```

Updating an Authentication Realm

Use the `set` subcommand to modify an existing authentication realm. You can only modify the realm's configuration properties.

The following example shows how to change the properties of a custom realm named `custom-db-realm`:

```
asadmin set configs.config.server-config.security-service.auth-realm.custom-
dbRealm.property.group-table=user_name_groups

asadmin set configs.config.server-config.security-service.auth-realm.custom-
dbRealm.property.group-name-column=db_group
```

WARNING	The realm's name and class name cannot be modified once its created.

Delete an Authentication Realm

Use the `delete-auth-realm` subcommand in remote mode to delete an existing authentication realm.

1. Ensure that the server is running,
2. Delete the realm by using the `delete-auth-realm` subcommand.

This example deletes the security realm named `custom-dbRealm`.

```
asadmin delete-auth-realm custom-dbRealm

Command delete-auth-realm executed successfully.
```

To Configure a JDBC or Digest Authentication Realm

Payara Server enables you to specify a database user's credentials (username and password) in the JDBC realm settings directly instead of in the settings of a JDBC resource and its corresponding connection pool. Using the `jdbc` type realm instead of the connection pool prevents other applications from browsing the database tables for user credentials.

TIP	The user password credential should be stored using a password alias for better security.

1. Create the database tables in which to store user credentials for the realm.

   How you create the database tables depends on the database that you are using.

2. Add user credentials to the database tables that you created.

   How you add user credentials to the database tables depends on the database that you are using.

3. Create a JDBC connection pool for the database.

   See "to-create-a-jdbc-connection-pool" in the Payara Server General Administration section.

4. Create a JDBC resource for the database.

   See "to-create-a-jdbc-resource" in the Payara Server General Administration section.

5. Create a new JDBC security realm.

WARNING	The JAAS context should be `jdbcDigestRealm` for digest authentication or `jdbcRealm` for other authentication types.

6. Modify the corresponding deployment descriptor that is associated with your application to specify the `jdbc` realm.

   - For an enterprise application in an Enterprise Archive (EAR) file, modify the `payara-application.xml` file.
   - For a web application in a Web Application Archive (WAR) file, modify the `web.xml` file.
   - For an enterprise bean in an EJB JAR file, modify the `payara-ejb-jar.xml` file.

     For more information about how to specify a security realm, see "how-to-configure-a-realm" in the Payara Server Application Development section.

7. Assign security roles to users in the realm.

   To assign a security role to a user, add a `security-role-mapping` element to the deployment descriptor that you modified.

This example shows a `security-role-mapping` element that assigns the security role `Employee` to a user account named `Calvin`:

```
<security-role-mapping>
    <role-name>Employee</role-name>
    <principal-name>Calvin</principal-name>
  </security-role-mapping>
```

To Enable LDAP Authentication on the Payara Server DAS

This procedure explains how to enable LDAP authentication to account for user logins

to the Payara Server Domain Administration Server (DAS). Login in to the DAS is typically only performed by Payara Server administrators who want to use the Administration Console or the Asadmin CLI.

NOTE	Ensure that you have properly configured an LDAP server that will host the user credentials with the administrators that will log in to the server's DAS.

Use the `asadmin configure-ldap-for-admin` subcommand to enable user authentication to the Payara Server DAS.

The command uses following syntax:

```
asadmin configure-ldap-for-admin --basedn "dn-list" --url [ldap|ldaps]://ldap-url
--ldap-group group-name
```

Where the arguments are defined as follows:

dn-list: LDAP BaseDN parameters **ldap-url**: URL and port number for the LDAP server; can use standard (`ldap`) or secure (`ldaps`) protocol **group-name**: LDAP group name for allowed users, as defined on the LDAP server.

For example:

```
asadmin configure-ldap-for-admin --basedn "dc=fish,dc=payara" \
--url ldap://payara.fish:3060 --ldap-group das-admins

asadmin configure-ldap-for-admin --basedn "dc=fish,dc=payara" \
--url ldaps://payara.fish:7501 --ldap-group das-admins
```

IMPORTANT	Be careful when running this command as once LDAP access is set up in this manner, it will prove challenging to revert this configuration, so it is encouraged to properly test these settings before running them in a production environment.

Administering File Users

When using a file security realm, as its administrator, you are responsible for integrating

users into the Payara Server environment so that their credentials are securely established, and they can access the applications and services that they are entitled to use.

NOTE	The file security realm is a good solution to consider when the user base of a business domain is sufficiently small and user identity administration doesn't require a complex setup. However, for most production environments it is recommended to use a robust solution using either an LDAP or relational database storage.

To Create a File User

Use the `create-file-user` subcommand in remote mode to create a new user by adding a new entry to the `keyfile`. The entry includes the username, password, and any groups that the user belongs to.

TIP	Multiple groups can be specified by using a colon(`:`) separated string.

NOTE	If secure administration is enabled as described in running-secure-admin, you cannot create an administrative user using a blank password.

Creating a new `file` realm user is a dynamic event and does not require server restart.

1. Ensure that the server is running.
2. Review the current groups of the security realm by using the "`list-file-groups`" subcommand.
3. Create a file user by using the "`create-file-user`" subcommand.

This example create user `jennifer` on the default realm `file` (no groups are specified).

The `asadmin --passwordfile` option specifies the name of a file that contains the password entries in a specific format. The entry for a password must have the `AS_ADMIN_` prefix followed by the password name in uppercase letters, an equals sign,

and the password. See the "`asadmin`" documentation for more information.

```
asadmin create-file-user --user admin --passwordfile=asadminpassword.txt jennifer

Command create-file-user executed successfully.
```

To List File Users

Use the `list-file-users` subcommand in remote mode to list the users that are in the `keyfile`.

This example lists file users on the default `file` realm file:

```
asadmin list-file-users

jennifer

Command list-file-users executed successfully.
```

To List File Groups

A group is a category of users classified by common traits, such as job title or customer profile. For example, users of an e-commerce application might belong to the `customer` group, and the big spenders might also belong to the `preferred` group. Categorizing users into groups makes it easier to control the access of large numbers of users.

A group is different from a role in that a role defines a function in an application, while a group is a set of users who are related in some way. For example, in the personnel application there might be groups such as `full-time`, `part-time`, and `on-leave`.

Users in these groups are all employees (the `employee` role). In addition, each user has its own designation that defines an additional level of employment.

Use the `list-file-groups` subcommand in remote mode to list groups for a specific user, or all file groups if the `--name` option is not specified.

This example lists the groups for user `joe`.

```
asadmin> list-file-groups --name joe
```

```
staff
manager

Command list-file-groups executed successfully
```

To Update a File User

Use the `update-file-user` subcommand in remote mode to modify the information for a specified user.

NOTE	If secure administration is enabled as described in "Running Secure Admin", you cannot update an administrative user to have a blank password.

1. Ensure that the server is running.
2. Update the user information by using the "`update-file-user`" subcommand.

The following command updates the groups for user `jennifer`:

```
asadmin update-file-user --passwordfile asadminpassword.txt --groups
staff:manager:engineer Jennifer

Command update-file-user executed successfully.
```

To Delete a File User

Use the `delete-file-user` subcommand in remote mode to remove a user entry from the `keyfile` by specifying the username.

1. Ensure that the server is running.
2. List users by using the "`list-file-users`" subcommand.
3. Delete the user by using the "`delete-file-user`" subcommand.

This example deletes user `jennifer` from the default `file` realm.

```
asadmin> delete-file-user jennifer

Command delete-file-user executed successfully.
```

Administering Message Security

This chapter provides information and procedures on configuring the message layer security for SOAP web services in a Payara Server environment.

NOTE	Message security is supported only in the Full Profile of Payara Server, not in the Web Profile.

Materials in this chapter assumes a basic understanding of security and web services concepts. For more information about security, see about-system-security-in-payara-server.

About Message Security in Payara Server

Message security enables a server to perform end-to-end authentication of web service invocations and responses at the message layer. Security information is inserted into messages so that it travels through the networking layers and arrives with the intact message at the message destination(s). Message security differs from transport layer security in that message security can be used to decouple message protection from message transport so that messages remain protected after transmission.

SOAP Web services deployed on Payara Server are secured by binding SOAP layer message security providers and message protection policies to the containers in which the applications are deployed, or to web service endpoints served by the applications. SOAP layer message security functionality is configured in the client-side containers of Payara Server by binding SOAP layer message security providers and message protection policies to the client containers or to the portable service references declared by client applications.

Message-level security can be configured for the entire Payara Server or for specific applications or methods. Configuring message security at the application level is discussed in the "Application Development section".

Security Tokens and Security Mechanisms

WS-Security is a specification that provides a communications protocol for applying

security to web services. The security mechanisms implement the specification. Web Services Interoperability Technologies (WSIT) implements WS-Security to provide interoperable message content integrity and confidentiality, even when messages pass through intermediary nodes before reaching their destination endpoint.

NOTE	WS-Security as provided by WSIT is in addition to existing transport-level security, which can still be used.

The Simple Object Access Protocol (SOAP) layer message security providers installed with Payara Server can be used to employ username/password and X.509 certificate security tokens to authenticate and encrypt SOAP web services messages.

Username Tokens: Payara Server uses username tokens in SOAP messages to authenticate the message sender. The recipient of a message containing a username token (within embedded password) validates that the message sender is authorized to act as the user (identified in the token) by confirming that the sender knows the password of the user.

+ When using a username token, a valid user database must be configured on Payara Server.

Digital Signatures: Payara Server uses XML digital signatures to bind an authentication identity to message content. Clients use digital signatures to establish their caller identity. Digital signatures are verified by the message receiver to authenticate the source of the message content (which might be different from the sender of the message.)

+ When using digital signatures, valid keystore and truststore files must be configured on Payara Server.

Encryption: The purpose of encryption is to modify the data so that it can only be understood by its intended audience. This is accomplished by substituting an encrypted element for the original content. When based on public key cryptography, encryption can be used to establish the identity of the parties who are authorized to read a message.

+ When using encryption, a Java Cryptography Extension (JCE) provider that supports the corresponding encryption algorithm suites must be installed.

Authentication Providers

The authentication layer is the message layer on which authentication processing must be performed. Payara Server enforces web services message security at the SOAP layer. The types of authentication that are supported include the following:

- Sender authentication, including username-password authentication
- Content authentication, including XML digital signatures

Payara Server invokes authentication providers to process SOAP message layer security. The message security providers provide information such as the type of authentication that is required for the request and response messages. The following message security providers are included with Payara Server:

- Client-side Provider: A client-side provider establishes (by signature or username/password) the source identity of request messages and/or protects (by encryption) request messages such that they can only be viewed by their intended recipients. A client-side provider also establishes its container as an authorized recipient of a received response (by successfully decrypting it) and validates passwords or signatures in the response to authenticate the source identity associated with the response. Client-side providers configured in Payara Server can be used to protect the request messages sent and the response messages received by server-side components (servlets and EJB components) acting as clients of other services.

  The default client provider is used to identify the client—side provider to be invoked for any application for which a specific client provider has not been bound.

- Server-side Provider: A server-side provider establishes its container as an authorized recipient of a received request (by successfully decrypting it), and validates passwords or signatures in the request to authenticate the source identity associated

with the request. A server-side provider also establishes (by signature or username/password) the source identity of response messages and/or protects (by encryption) response messages such that they can only be viewed by their intended recipients. Server-side providers are only invoked by server-side containers.

The default server provider is used to identify the server—side provider to be invoked for any application for which a specific server provider has not been bound.

Message Protection Policies

A request policy defines the authentication policy requirements associated with request processing performed by the authentication provider. Policies are expressed in message sender order such that a requirement that encryption occur after content would mean that the message receiver would expect to decrypt the message before validating the signature. The response policy defines the authentication policy requirements associated with response processing performed by the authentication provider.

Message protection policies are defined for request message processing and response message processing. The policies are expressed in terms of requirements for source and/or recipient authentication. The providers apply specific message security mechanisms to cause the message protection policies to be realized in the context of SOAP web services messages.

- Source Authentication Policy: A source authentication policy represents a requirement that the identity of the entity that sent a message or that defined the content of a message be established in the message such that it can be authenticated by the message receiver.
- Recipient Authentication Policy. A recipient authentication policy represents a requirement that the message be sent such that the identity of the entities that can receive the message can be established by the message sender.

Request and response message protection policies are defined when a security provider is configured into a container. Application-specific message protection policies (at the

granularity of the web service port or operation) can also be configured within the Payara Server deployment descriptors of the application or application client. In any situation where message protection policies are defined, the request and response message protection policies of the client must be equivalent t) the request and response message protection policies of the server.

For more information about defining application-specific message protection policies, see "securing-applications" in the Application Development section.

Application-Specific Web Services Security

Application-specific web services security functionality is configured (at application assembly) by defining the `message-security-binding` elements in the Payara Server deployment descriptors of the application. These `message-security-binding` elements are used to associate a specific security provider or message protection policy with a web service endpoint or service reference, and might be qualified so that they apply to a specific port or method of the corresponding endpoint or referenced service.

For information about defining application-specific message protection policies, see "securing-applications" in the Application Development section.

Message Security Administration

When Payara Server is installed or a new domain is created, SOAP layer message security providers are configured in the client and server-side containers of Payara Server, where they are available for binding for use by the containers, or by individual applications or clients deployed in the containers.

Default providers are configured with a simple message protection policy that, if bound to a container, or to an application or client in a container, would cause the source of the content in all request and response messages to be authenticated by XML digital signature.

Payara Server administrative interfaces can be used as follows:

- To modify the message protection policies enforced by the providers
- To bind the existing providers for use by the server-side containers of Payara Server
- To create new security provider configurations with alternative message protection policies

Analogous administrative operations can be performed on the SOAP message layer security configuration of the application client container. If you want web services security to protect all web services applications deployed on Payara Server, see enabling-message-security-for-application-clients.

Message Security Tasks

The general implementation tasks for message security include some or all of the following:

1. If you are using a username token, verifying that a user database is configured for an appropriate security realm

NOTE	When using a username/password token, an appropriate realm must be configured and a user database must be configured for the realm.

2. Managing certificates and private keys, if necessary
3. Enabling the Payara Server default providers
4. Configuring new message security providers

Message Security Roles

In a traditional Payara Server management infrastructure, the administrator and the application deployer are expected to take primary responsibility for configuring message security.

System Administrator

The system administrator is responsible for the following message security tasks:

- Administering server security settings and certificate databases

- Administering keystore and truststore files
- Configuring message security providers on Payara Server
- Turning on message security

Application Deployer

The application deployer is responsible for the following message security tasks:

- Specifying (at application reassembly) any required application-specific message protection policies if such policies have not already been specified by the developer/assembler.
- Modifying Payara Server deployment descriptors to specify application-specific message protection policies information (message-security-binding elements) to web service endpoint and service references.

Application Developer

The application developer is responsible for the following message security tasks:

- Determining if an application-specific message protection policy is required by the application

  If so, the developer ensures that the required policy is specified at application assembly time.

- Specifying how SOAP web services should be set up for message security.

  NOTE | Message security can be set up by the administrator so that all web services are secured, or by the application deployer when the security provider or protection policy bound to the application must be different from that bound to the container.

- Turning on message security if authorized to do so by the administrator.

Enabling Default Message Security Providers for SOAP Web Services

By default, message security is disabled on Payara Server. Default message security

providers have been created, but are not active until you enable them. After the providers have been enabled, message security is enabled.

To Enable a Default Server Provider

To enable message security for web services endpoints deployed in Payara Server, you must specify a security provider to be used by default on the server side. If you enable a default provider for message security, you also need to enable providers to be used by clients of the web services deployed in Payara Server.

1. Specify the default server provider by using the set subcommand.

   Use the following syntax:

```
asadmin set --port admin-port server-config.security-service.message-security-
config.SOAP.default_provider=ServerProvider
```

2. To apply your changes to applications that are already running, restart the server instance.

To Enable a Default Client Provider

To enable message security for web service invocations originating from deployed endpoints, you must specify a default client provider. If you enabled a default client provider for Payara Server, you must ensure that any services invoked from endpoints deployed in Payara Server are compatibly configured for message layer security.

1. Specify the default client provider by using the set subcommand.

   Use the following syntax:

```
asadmin set --port admin-port server-config.security-service.message-security-
config.SOAP.default_client_provider=ClientProvider
```

2. To apply your changes to applications that are already running, restart the server instance.

Configuring Message Protection Policies

Message protection policies are defined for request message processing and response message processing. The policies are expressed in terms of requirements for source and/or recipient authentication.

The providers apply specific message security mechanisms to cause the message protection policies to be realized in the context of SOAP web services messages.

Message Protection Policy Mapping

The following table shows message protection policy configurations and the resulting message security operations performed by the WS-Security SOAP message security providers for that configuration.

Table 160. Message Protection Policy Mapping to WS-Security SOAP Operations

Message Protection Policy	Resulting WS-Security SOAP Message Protection Operations
`auth-source="sender"`	The message contains a `wsse:Security` header that contains a `wsse:UsernameToken` (with password).
`auth-source="content"`	The content of the SOAP message Body is signed. The message contains a `wsse:Security` header that contains the message body signature represented as a `ds`:`Signature`.

`auth-source="sender"` `auth-recipient="before-content"`
 	
OR
 	
`auth-recipient="after-content"`
	The content of the SOAP message Body is encrypted and replaced with the resulting `xend:EncryptedData`.
	
	The message contains `a wsse:Security` header that contains a `wsse:UsernameToken (with password)` and an `xenc:EncryptedKey`.
	
	The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.

Message Protection Policy	Resulting WS-Security SOAP Message Protection Operations
`auth-source="content"` `auth-recipient="before-content"`	The content of the SOAP message body is encrypted and replaced with the resulting `xend:EncryptedData`. The `xenc:EncryptedData` is signed.The message contains `a` `wsse:Security` header that contains an `xenc:EncryptedKey` and a `ds`:`Signature`. The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.
`auth-source="content"` `auth-recipient="after-content"`	The content of the SOAP message Body is signed, then encrypted, and then replaced with the resulting `xend:EncryptedData`. The message contains a `wsse:Security` header that contains an `xenc:EncryptedKey` and a `ds:Signature`. The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.

`auth-recipient="before-content"`
 	
OR
 	
`auth-recipient="after-content"`
	The content of the SOAP message Body is encrypted and replaced with the resulting `xend:EncryptedData`.
	
	The message contains `a wsse:Security` header that contains an `xenc:EncryptedKey`.
	
	The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.

No policy specified.	No security operations are performed by the modules.

To Configure the Message Protection Policies for a Provider

Typically, you would not reconfigure a provider. However, if needed for your situation, you can modify a provider's message protection policies by changing provider type, implementation class, and provider-specific configuration properties. To understand the results of different combinations, see the table in the above section.

Use the "`set`" subcommand to set the response policy, then replace the word `request` in the following commands with the word `response`.

1. Add a request policy to the client and set the authentication source by using the set subcommand.

   For example:

   ```
   asadmin set server-config.security-service.message-security-
   config.SOAP.provider-config.ClientProvider.request-policy.auth_source=[sender |
   content]
   ```

2. Add a request policy to the server and set the authentication source by using the `set` subcommand.

   For example:

   ```
   asadmin set server-config.security-service.message-security-
   config.SOAP.provider-config.ServerProvider.request-policy.auth_source=[sender |
   content]
   ```

3. Add a request policy to the client and set the authentication recipient by using the `set` subcommand:

   For example:

   ```
   asadmin set server-config.security-service.message-security-
   config.SOAP.provider-config.ClientProvider.request-policy.auth_recipient
   =[before-content | after-content]
   ```

4. Add a request policy to the server and set the authentication recipient by using the `set` subcommand:

   For example:

   ```
   asadmin set server-config.security-service.message-security-
   config.SOAP.provider-config.ServerProvider.request-policy.auth_recipient
   =[before-content | after-content]
   ```

Setting the Request and Response Policy for the Application Client Configuration

The request and response policies define the authentication policy requirements associated with request and response processing performed by the authentication provider.

Policies are expressed in message sender order such that a requirement that encryption occur after content would mean that the message receiver would expect to decrypt the message before validating the signature.

To achieve message security, the request and response policies must be enabled on both the server and client. When configuring the policies on the client and server, make sure that the client policy matches the server policy for request/response protection at application-level message binding.

To set the request policy for the application client configuration, modify the Payara Server-specific configuration for the application client container as described in enabling-message-security-for-application-clients.

Message Security Policy Setting Example

In the application client configuration file, the `request-policy` and `response-policy` elements are used to set the request policy, as shown in the following code snippet.

NOTE | Additional code in the snippet is provided as illustration and might differ slightly in your environment.

```
<client-container>
  <target-server name="your-host" address="your-host"  port="your-port"/>
  <log-service file="" level="WARNING"/>
  <message-security-config auth-layer="SOAP" default-client-
provider="ClientProvider">
    <provider-config class-
name="com.sun.enterprise.security.jauth.ClientAuthModule" provider-
id="clientprovider" provider-type="client">
      <request-policy auth-source="sender | content" auth-recipient="after-content
| before-content"/>
      <response-policy auth-source="sender | content" auth-recipient="after-content
| before-content"/>
       <property name="security.config" value="as-install/lib/appclient/wss-client-
config.xml"/>
```

```
    </provider-config>
  </message-security-config>
</client-container>
```

Valid values for `auth-source` include `sender` and `content`. Valid values for `auth-recipient` include `before-content` and `after-content`. A table describing the results of various combinations of these values can be found in configuring-message-protection-policies.

To not specify a request or response policy, leave the element with no content:

```
<response-policy/>
```

Administering Non-Default Message Security Providers

To Create a Message Security Provider

Use the `create-message-security-provider` subcommand in remote mode to create a new message provider for the security service. If the message layer does not exist, the message layer is created, and the provider is created under it.

1. Ensure that the server is running.
2. Create the message security provider by using the create-message-security-provider subcommand.
3. If needed, restart the server, as some properties require server restart.

This example creates the new message security provider name `mySecurityProvider`:

```
asadmin create-message-security-provider --classname
com.sun.enterprise.security.jauth.ClientAuthModule --providertype client
mySecurityProvider

Command create-message-security-provider executed successfully.
```

To List Message Security Providers

Use the `list-message-security-providers` subcommand in remote mode to list the message providers for the security layer.

1. Ensure that the server is running.
2. List the message security providers by using the list-message-security-providers subcommand.

This example lists the message security providers for a message layer.

```
asadmin list-message-security-providers --layer SOAP

XWS_ClientProvider
ClientProvider
XWS_ServerProvider
ServerProvider

Command list-message-security-providers executed successfully.
```

To Update a Message Security Provider

1. Ensure that the server is running.
2. List the message security providers by using the list-message-security-providers subcommand.
3. Modify the values for the specified message security provider by using the "`set`" subcommand.

   The message security provider is identified by its **dotted name**.

To Delete a Message Security Provider

Use the `delete-message-security-provider` subcommand in remote mode to remove a message security provider.

1. Ensure that the server is running.
2. List the message security providers by using the "`list-message-security-providers`" subcommand.
3. Delete the message security provider by using the delete-message-security-provider subcommand.

This example deletes the `myServerityProvider` message security provider.

```
asadmin delete-message-security-provider --layer SOAP myServerityProvider

Command delete-message-security-provider executed successfully.
```

To Configure a Servlet Layer Server Authentication Module (SAM)

You can configure a Jakarta Authentication Server Authentication Module (SAM) as an HTTP Servlet-layer message security provider, either through the Administration Console or with the `create-message-security-provider` subcommand.

1. Ensure that the server is running.
2. Create the message security provider by using the create-message-security-provider subcommand.
3. Bind the message security provider for use with your application.

   You do this by defining the `httpservlet-security-provider` attribute in the `glassfish-web.xml` (deprecated) or `payara-web.xml` deployment descriptor corresponding to your application.

   Set the value of the attribute to the provider name you assigned to the message security provider. For example, if you use MySAM when you create the message security provider the entry would be `httpservlet-security-provider="MySAM"`.
4. Restart the server instance.

This example creates a new message security provider named `mySAM`.

```
asadmin create-message-security-provider --layer=HttpServlet --classname
com.sun.glassfish.oamsam.OAMAuthenticatorSAM --providertype server --property
oam.resource.hostid.variation="your-host-system.com" mySAM

Creation of message security provider named mySAM completed successfully

Command create-message-security-provider executed successfully.
```

The subcommand results in the following `domain.xml` configuration entry:

```
<message-security-config auth-layer="HttpServlet">
```

```
    <provider-config provider-type="server" provider-id="mysam"  class-
name="com.sun.glassfish.oamsam.OAMAuthenticatorSAM">
        <property name="oam.resource.hostid.variation" value="your-host-
system.com"></property>
        <request-policy></request-policy>
        <response-policy></response-policy>
    </provider-config>
</message-security-config>
```

To list the HttpServlet message security providers, use the `list-message-security-providers` subcommand:

```
asadmin list-message-security-providers --layer HttpServlet

list-message-security-providers successful
GFConsoleAuthModule
mySAM

Command list-message-security-providers executed successfully.
```

Enabling Message Security for Application Clients

The message protection policies of client providers must be configured such that they are equivalent to the message protection policies of the server-side providers they will be interacting with. This is already the situation for the providers configured (but not enabled) when Payara Server is installed or a new domain is created.

To enable message security for client applications, modify the Payara Server specific configuration for the application client container. The process is analogous to the process in configuring-message-protection-policies.

Administering Security in a High-Availability Environment

This chapter describes important information about administering security in a high availability environment (using traditional clusters or deployment groups).

This chapter assumes that you are familiar with security features such as authentication, authorization, and certificates. If you are not, see administering-system-security.

Configuring Certificates in Cluster Mode

The sections certificates-and-secure-transport and "Administering JSSE Certificates"

describe the relevant concepts and use of certificates in Payara Server.

By default, Payara Server uses self-signed certificates. The self-signed certificates that Payara Server uses will not be trusted by clients by default because a certificate authority does not vouch for the authenticity of the certificate.

You can instead use your own certificates, as described in "Using Your Own Certificates".

Dynamic Reconfiguration

Administrative commands that you execute on the domain administration server (DAS) must either be replicated on the affected server instances, or on all server instances that are part of a cluster or deployment group.

Payara Server replicates the commands by sending the same administration command request that was sent to the DAS to the server instances. As a result of replicating the commands on the DAS and the individual instances, the DAS and the instances make the same changes to their respective copies of the domain's configuration.

NOTE	It is recommended that you enable secure admin as described in managing-administrative-security so that Payara Server securely transfers these files on the network.

Dynamic reconfiguration refers to using the `--target` operand to CLI subcommands to make a change to a server instance (if the user-specified target is a server instance), or all server instances that are part of a cluster or deployment group (if the user-specified target is a cluster or deployment group).

For example, in the following command:

```
asadmin create-jdbc-resource some-options --target some-target
```

The `--target` operand allows the following values:

- `server` - Performs the command on the default server instance. This is the default value.
- `configuration_name` - Performs the command in the specified configuration.
- `cluster_name` - Performs the command on all server instances in the specified cluster.
- `deployment_group_name` - Performs the command on all server instances in the specified deployment group.
- `instance_name` - Performs the command on a specified server instance.

If a command fails while targeting a cluster or deployment group, the status shows all server instances where dynamic reconfiguration failed, and suggests corrective measures to make progress.

The command status also shows when a restart is required for each server instance.

Enabling Dynamic Configuration

Dynamic configuration is enabled by default and no additional action is required. Use the following command to enable dynamic configuration from the command line:

```
asadmin --user user --passwordfile password-file set <cluster-name-config>.dynamic-reconfiguration-enabled=true
```

To enable dynamic configuration from the Administration Console, perform the following steps:

1. Expand the *Configurations* node.
2. Click the name of the cluster's configuration or a shared configuration object used by the instances of a deployment group.
3. On the *System Properties* page, check the *Dynamic Reconfiguration* Enabled box.
4. Save your changes.

IMPORTANT	The dynamic reconfiguration feature applies only to server

> instances, not the DAS. Therefore, you cannot "disable" dynamic reconfiguration on the DAS.
> Some commands and changes done in the DAS may be applied dynamically on the DAS, so consult the documentation on the command reference to find out if they support dynamic re-configuration on the DAS/

Understanding Synchronization

As described in "resynchronizing-payara-server-instances-and-the-das" in the High Availability section, configuration data for a Payara Server instance is stored in the repository of the DAS and in a cache on the host that is local to the instance.

The configuration data in these locations must be synchronized. The cache is synchronized only when a user uses the administration tools to start or restart an instance.

Managing Administrative Security

This chapter describes how to manage administrative security by using Payara Server's secure administration feature.

This chapter assumes that you are familiar with security features such as authentication, authorization, and certificates. If you are not, first see administering-system-security.

Secure Administration Overview

The secure administration feature allows an administrator to secure all administrative communication between the domain administration server (DAS), any remote instances, and administration clients such as the `asadmin` CLI utility, the administration console's web interface, and REST clients using the Management API.

In addition, secure administration helps to prevent DAS-to-DAS and instance-to-instance traffic, and carefully restricts administration-client-to-instance traffic.

The secure administration feature, which is henceforth referred to as **secure admin**,

provides a secure environment, in which you can be confident that rogue users or processes cannot intercept or corrupt administration traffic or impersonate legitimate Payara Server components.

When you install Payara Server or create a new domain, secure admin is disabled by default. When secure admin is disabled, Payara Server does not encrypt administrative communication among the system components, but it **does not accept administrative connections from remote hosts**.

The following subcommands enable and disable secure admin:

`enable-secure-admin`: The `enable-secure-admin` subcommand turns on secure admin. Payara Server uses SSL/TLS encryption to protect subsequent administrative traffic and will ALSO accept remote administrative connections. Enabling secure admin affects the entire domain, including the DAS and all of its instances. The DAS must be running to run `enable-secure-admin`.

+ You must restart the DAS immediately after enabling secure admin, and then start any instances you want to run.

+ CAUTION: The `enable-secure-admin` subcommand fails if ANY administrator users have an empty password credential assigned. Before retrying the command, ALL administrator users should have non-empty passwords assigned.

`disable-secure-admin`: The `disable-secure-admin` subcommand turns off secure admin. Payara Server no longer encrypts administrative messages and will no longer accept remote administration connections. Disabling secure admin affects the entire domain, including the DAS and all instances. The DAS must be running to run `disable-secure-admin`.

+ You must restart the DAS immediately after disabling secure admin, and then start any instances you want to run.

+ If secure admin is not enabled, this subcommand has no effect.

This section describes how to use these commands to run secure admin, and the implications of doing so.

How Secure Admin Works: the Big Picture

Secure admin is a domain-wide setting. It affects the DAS and all instances and all administration clients.

Functions Performed by Secure Admin

The `enable-secure-admin` subcommand performs the following functions. Subsequent sections describe these functions in more detail.

- Enables the secure admin behavior, optionally setting which **aliases** are to be used for identifying the DAS and instance certificates.
- Adjusts all configurations in the domain, including the `default-config` template configuration.
- Adjusts HTTP settings:
  - SSL/TLS is enabled in the DAS' admin listener and the instances' admin listeners configurations.
  - Port unification (that is, HTTP and HTTPS are handled by the same port), http—to—https redirection, and client authentication (`client-auth=want`) are enabled.
  - Configures SSL to use the administration truststore.
  - Configures SSL to use the administration keystore and the correct alias (for the self-signed cert) for authenticating itself. (You can use a CA-issued certificate instead, as described in using-your-own-certificates.

NOTE	HTTP configuration on the DAS and each instance is identical, with the exception that the DAS uses the `s1as` alias for SSL/TLS authentication and the instances use the `glassfish-instance` alias by default.

A server restart is required to apply the aforementioned changes. The restart also synchronizes the restarted instances.

Which Administration Account Is Used?

If only one administration account exists in the default `admin` realm, Payara Server treats that account as the current default administration account. In this case, when you run an `asadmin` command, you do not need to specify a username. If a password for that username is required, you need to specify it, typically by using the `--passwordfile` option or by letting `asadmin` prompt you for it.

NOTE	By default, Payara Server includes a single account for user `admin` which has an empty password.

If multiple admin accounts exist, then Payara Server does not recognize any administrator username as the default. You must then specify a valid username via the `--user` option when you use the `asadmin` command (or by or defining the `AS_ASDMIN_USER` environment variable), and its associated password (if it is not empty).

The username and password used for a login attempt must match the username and password (if required) for an account defined in the `admin` realm, and you must have set up the account as a member of the `admin` group.

What Authentication Methods Are Used for Secure Administration?

The secure admin feature enforces security via the following authentication methods:

- The DAS and instances authenticate to each other via mutual (two-way) SSL/TLS certificate authentication. The DAS authenticates to clients via one-way SSL/TLS certificate authentication.

NOTE	The domain creation process creates a default keystore and truststore, plus a default private key for the DAS. Secure admin uses this initial configuration to set up the default truststore so that the

> DAS and instances always trust each other.

- Remote administration clients (the Asadmin CLI, the web administration console, and IDE extension plugins) must accept the public certificate presented by the DAS. If accepted, remote administration clients then send a username and password (via HTTP `Basic` authentication). The receiving DAS or instance makes sure those credentials are valid in its configured `admin` realm, and authenticates and authorizes the user.
- A locally-running Asadmin CLI (that is, connecting to an instance on the same host) authenticates and authorizes to the co-located instance using a locally-provisioned password.
- Credentials or other sensitive information sent over the network are always encrypted if secure admin is enabled. No credentials are sent in the clear if secure admin is enabled. (If secure admin is disabled, credentials are sent in clear text.)

  Messages between administration clients and the DAS, between the DAS and remote instances, and between local administration clients and instances are encrypted using SSL/TLS. This is true even if you explicitly set the `asadmin --secure` option to `false`.

The following table shows which authentication methods are employed when secure admin is enabled or disabled.

Table 161. Authentication Methods Employed

Access Method	Secure Admin is Disabled	Secure Admin is Enabled
Remote administration access to the DAS	Rejected.	Username/password authentication. (Client must also accept server certificate.)
Communication between DAS and instances	Cleartext messages. No mutual authentication.	SSL-encrypted messages. SSL mutual authentication using certificates.

Access Method	Secure Admin is Disabled	Secure Admin is Enabled
Communication between administration clients and DAS	Cleartext messages. No DAS authentication.	SSL-encrypted messages. DAS uses SSL certificate server authentication.
Local asadmin client to instance on same node	Cleartext messages. Locally-provisioned password mechanism is used.	SSL-encrypted messages. Locally-provisioned password mechanism is used.

Understanding How Certificate Authentication Is Performed

The domain creation process creates a primary (private) key and a self-signed certificate for the DAS, and a separate private key and self-signed certificate for remote instances.

Then, when you enable secure admin, the following actions are performed:

- Both private keys are stored in the domain-wide DAS keystore file, `keystore.p12`.
- Both public certificates are stored in the domain-wide DAS truststore file, `cacerts.p12`.

When the DAS sends a message to an instance:

1. The instance asks the DAS to provide an SSL/TLS certificate.
2. The DAS sends the certificate with the alias you specified using the `--adminalias` option when you ran the `enable-secure-admin` subcommand.
3. The instance makes sure the certificate is valid and Payara Server makes sure that the security `Principal` associated with the incoming request (provided automatically by the HTTP engine and the SSL/TLS implementation provided by the Java runtime) matches the `Principal` associated with the`adminalias` from the instance's truststore.

What Certificates Are Used?

When you enable secure admin, you can optionally set the `--adminalias` and `--instancealias` options that tell secure admin which aliases to use for the DAS and

instance certificates.

The DAS uses the alias associated with the `--instancealias` option to check incoming requests that use SSL/TLS cert authentication. Conversely, instances use the alias associated with the `--adminalias` option to check incoming requests with certificate authentication.

By default, `--adminalias` of the `enable-secure-admin` subcommand uses the `s1as` alias, and the `--instancealias` option uses the `glassfish-instance` alias, both of which identify the default self-signed certificates.

You can use your tool of choice, such as `keytool`, to list the default self-signed certificates in the keystore, similar to the following:

```
keytool -list -keystore keystore.p12

Enter keystore password:

Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 2 entries

glassfish-instance, Jan 3, 2023, PrivateKeyEntry,
Certificate fingerprint (MD5): 06:A4:83:84:57:52:9C:2F:E1:FD:08:68:BB:2D:ED:E8
s1as, Jan 3, 2023, PrivateKeyEntry,
Certificate fingerprint (MD5): 8B:7D:5A:4A:32:36:1B:5D:6A:29:66:01:B0:A3:CB:85
```

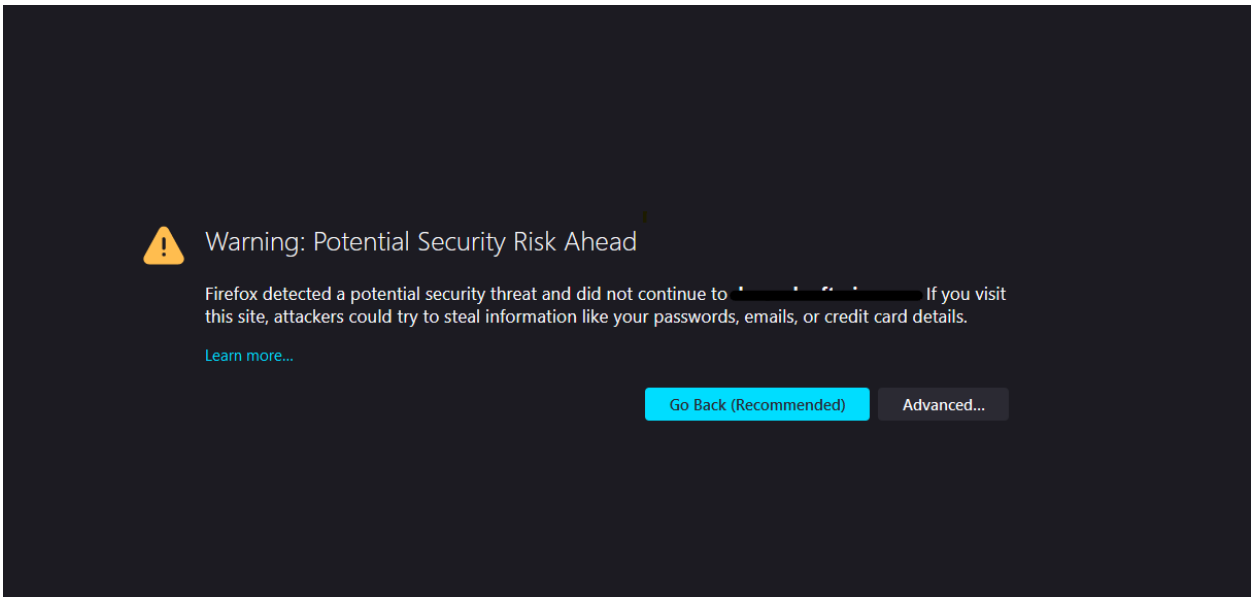
The `--adminalias` and `--instancealias` values are maintained. Because of this design, normal instance creation operations (create-instance over SSH and create-local-instance) apply the up-to-date keystore, truststore, and configuration to each instance.

Self-Signed Certificates and Trust

The self-signed certificates that Payara Server uses will not be trusted by clients by default because a certificate authority (CA)) does not vouch for the authenticity of the certificate.

If you enable secure admin and then contact the DAS using an administration client, that client will detect whether the certificate is automatically trusted. Most modern web browsers will deny access to the web interface of the Admin console and reject any

communication until the user explicitly creates an exception to continue. And example is shown in the following figure:



Similarly, the first time the Asadmin CLI receives an untrusted certificate, it displays the certificate and lets you accept it or reject it, as follows:

```
> asadmin enable-secure-admin
Command enable-secure-admin executed successfully.

> asadmin restart-domain domain1
...
Command start-domain executed successfully.

> asadmin list-applications
[
[
  Version: V3
  Subject: CN=sample.payara.fish,OU=Payara,O=Payara Foundation,L=Great Malvern,ST
=Worcestershire,C=UK
  Signature Algorithm: SHA1withRSA, OID = 1.2.840.113549.1.1.5

  Key:  Sun RSA public key, 1024 bits
  modulus:
91604359507378444963235875637429733088161806229854910107270225245885674079656358328
56880000154850721926291086431192482493819504582208856345925321638321100660819657204
75752389641560683347149956407122672247805640710231886279679746562450905199563763572
88295037519504394674686082145398885236913866246525691704749
  public exponent: 65537
  Validity: [From: Tue Jan 04 14:30:08 EST 2023,
              To: Fri Jan 01 14:30:08 EST 2033]
  Issuer: CN=sample.payara.fish,OU=Payara,O=Payara Foundation,L=Great Malvern,ST
=Worcestershire,C=UK
  SerialNumber: [    4d237540]
```

```
---------------------------------------------------------------------------------------------------
   Certificate Extensions: 1
   [1]: ObjectId: 2.5.29.14 Criticality=false
   SubjectKeyIdentifier [
   KeyIdentifier [
       0000: AF 8B 90 1E 51 9A 80 1B   EB A4 D9 C6 01 8A A0 FD  ....Q...........
       0010: DE EC 83 8A                                        ....
       ]
   ]

]
   Algorithm: [SHA1withRSA]
   Signature:
     0000: 3F 2B 30 CE 97 0B 5E F3   72 0E 60 18 8D 3B 04 DC  ?+0...^.r.\`..;..
     0010: 26 E6 7A 6F D0 19 CC 26   1D 90 C0 DE 33 4E 53 FB  &.zo...&....3NS.
     0020: DC E7 AE 78 9E BA EF 14   86 57 36 D4 3E 9B C9 FB  ...x.....W6.\>...
     0030: C0 B4 EF 72 27 D9 4F 79   1F 89 91 B8 96 26 33 64  ...r\'.Oy.....&3d
     0040: 9F 4B 04 4B 83 B9 BF 4D   54 B4 8F 75 17 1A 51 BD  .K.K...MT..u..Q.
     0050: F3 69 94 CE 90 95 08 55   2C 07 D2 23 AC AE EC 6D  .i.....U,..#...m
     0060: 84 B6 3D 00 FB FE 92 50   37 1A 2D 00 F1 21 5C E6  ..=....P7.-..!\.
     0070: 1F 39 26 B2 5D C1 FD C8   B1 4F CC EE 26 84 B8 B5  .9&.]....O..&...
]
Do you trust the above certificate [y|N] -->
```

CAUTION	When accepted for the first time, the certificate will be automatically accepted in subsequent Asadmin CLI operations.

The Asadmin CLI locally saves accepted certificates in the `.gf-client/truststore` file in the OS user's home directory. You do not generally need to work with the file directly, but if you delete or move the file, `asadmin` will prompt you again when it receives untrusted certificates.

Some `asadmin` commands can contact an instance directly to retrieve information (but not to make configuration changes). The instances do not use the same certificate as the DAS, so in these cases the command will also prompt you to accept or reject the specific instance certificate.

Using Your Own Certificates

By default, `--adminalias` of the `enable-secure-admin` subcommand uses the `s1as` alias, and the `--instancealias` option uses the `glassfish-instance` alias, both of which identify default self-signed certificates prepared by the domain at its creation.

You can instead have Payara Server use your own certificates for this purpose by first

importing your certificates to the keystore and truststore, and then enabling secure admin and specify the aliases for your custom certificates.

NOTE	In general, in most cases it is recommended to use custom certificates for the DAS as remote access is usually done on un-restricted environments.

It is also possible to use `s1as` and `glassfish-instance` as the alias names for your own certificates when importing them to the keystore and truststore. A benefit of doing so is that you would not have to specify alias names with the `enable-secure-admin` subcommand.

In addition, your own certificate identified by the `s1as` alias would be used in all other cases within the domain where the `s1as` alias is used (by default), such as in the SSL configuration of the IIOP and `http-listener-2` listeners, and as the `encryption.key.alias` and `signature.key.alias` used for provider configuration in the SOAP authentication layer for Message Security configuration.

You may find the wide-reaching effect of using the `s1as` alias with your own certificate to be either a useful feature or an unintended consequence. Therefore, you should understand the implications of using the `s1as` alias before doing so.

If you decide to use the `s1as` and `glassfish-instance` aliases with your own certificates, you will first need to disable secure admin (if enabled) and then change or delete the exiting `s1as` alias from both the `keystore.p12` keystore and `cacerts.p12` truststore for the DAS.

You can also use the `--changealias` or`--delete` option of `keytool` to accomplish this. After that import your own certificates.

IMPORTANT	When renewing an expired certificate (since most CA-issued certificates would have a lifetime of a year in general), you must first disable secure admin, replace the certificates in the keystore and truststore with the renewed certificates and then re-enable

> secure admin to correctly re-established secure administration to the domain and its instances.

An Alternate Approach: Using Distinguished Names to Specify Certificates

By default, the DAS uses the alias associated with the `--instancealias` option to check incoming requests that use SSL/TLS cert authentication. Conversely, instances use the alias associated with the `--adminalias` option to check incoming requests with certificate authentication.

The enable-secure-admin-principal subcommand provides an alternate approach. `enable-secure-admin-principal` instructs Payara Server to accept admin requests when accompanied by an SSL certificate with the specified distinguished name (DN).

> **NOTE** Any certificate you specify with `enable-secure-admin-principal` must either be issued by a trusted certificate authority or, if it is self-signed, must already be imported in the server's truststore.

For example, assume that you write your own admin client that uses the REST interface. When your client establishes the connection, it can choose which certificate to use for its client cert. You would then specify the DN of this certificate to `enable-secure-admin-principal`.

You must specify either the DN or the `--alias` option of the `enable-secure-admin-principal` subcommand.

If you specify the DN, Payara Server records the value you specify as the DN. You specify the DN as a comma-separated list in quotes. For example, `"CN=sample.payara.fish,OU=Payara,O=Payara Foundation,L=Great Malvern,ST=Worcestershire,C=UK"`.

> **NOTE** The `enable-secure-admin-principal` subcommand accepts the string you enter and does not immediately validate it. However, secure

> admin must be able to match the DN you specify in order to use it.

If you have sufficient privileges to view the content of the keystore, you can use keytool to display the DN of a certificate:

```
keytool -v -list -keystore keystore.p12

Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 2 entries

Alias name: glassfish-instance
Creation date: Jul 7, 2023
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=sample.payara.fish,OU=Payara,O=Payara Foundation,L=Great Malvern,ST
=Worcestershire,C=UK
Issuer: CN=sample.payara.fish,OU=Payara,O=Payara Foundation,L=Great Malvern,ST
=Worcestershire,C=UK
Serial number: 4e15d6e7
Valid from: Thu Jul 07 11:55:19 EDT 2023 until: Sun Jul 04 11:55:19 EDT 2033
Certificate fingerprints:
         MD5:  05:6E:01:D6:CE:9D:29:DA:55:D9:10:5E:BE:CC:55:05
         SHA1: 2A:6D:A2:52:A5:2B:ED:DE:CD:B4:76:4A:65:9D:B5:79:A6:EA:3C:10
         Signature algorithm name: SHA1withRSA
         Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
    KeyIdentifier [
    0000: 96 99 36 B6 CF 60 1E 8A   AE 25 75 4E C8 34 AA AB  ..6..`...%uN.4..0010:
E1 3B CF 03                                              .;..
    ]
]
```

If you use the `--alias aliasname` form, then Payara Server looks in its truststore for a certificate with the specified alias and uses the DN associated with that certificate. `aliasname` must be an alias associated with a certificate currently in the truststore.

If you have sufficient privileges to view the contents of the truststore, you can use keytool to display the alias of a certificate:

```
keytool -v -list -keystore cacerts.p12

Alias name: glassfish-instance
Creation date: Jul 7, 2023
```

```
Entry type: trustedCertEntry

Owner: CN=sample.payara.fish,OU=Payara,O=Payara Foundation,L=Great Malvern,ST
=Worcestershire,C=UK
Issuer: CN=sample.payara.fish,OU=Payara,O=Payara Foundation,L=Great Malvern,ST
=Worcestershire,C=UK
Serial number: 4e15d6e7
Valid from: Thu Jul 07 11:55:19 EDT 2023 until: Sun Jul 04 11:55:19 EDT 2033
Certificate fingerprints:
         MD5:  05:6E:01:D6:CE:9D:29:DA:55:D9:10:5E:BE:CC:55:05
         SHA1: 2A:6D:A2:52:A5:2B:ED:DE:CD:B4:76:4A:65:9D:B5:79:A6:EA:3C:10
         Signature algorithm name: SHA1withRSA
         Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
    KeyIdentifier [
    0000: 96 99 36 B6 CF 60 1E 8A   AE 25 75 4E C8 34 AA AB  ..6..`...%uN.4..
    0010: E1 3B CF 03                                        .;..
    ]
]
```

When you run `enable-secure-admin`, Payara Server automatically records the DNs for the admin alias and the instance alias, whether you specify those values or use the defaults. You do not need to run `enable-secure-admin-principal` yourself for those certificates.

Other than these certificates, you must run `enable-secure-admin-principal` for any other DN that Payara Server should authorize to send admin requests. This includes DNs corresponding to trusted certificates (those with a certificate chain to a trusted authority.)

You can run `enable-secure-admin-principal` multiple times so that Payara Server accepts admin requests from a client sending a certificate with any of the DNs you specify.

The following example shows how to specify a `DN` for authorizing access in secure administration:

```
asadmin enable-secure-admin-principal
"CN=sample.payara.fish,OU=Payara,O=Payara Foundation,L=Great
Malvern,ST=Worcestershire,C=UK"
```

```
Command enable-secure-admin-principal executed successfully.
```

You can use the disable-secure-admin-principal subcommand to disable a specific certificate for authenticating and authorizing access in secure admin. You must specify either the DN or the `--alias` option of the `disable-secure-admin-principal` subcommand. To disable multiple certificates for authenticating and authorizing access in secure admin, run the `disable-secure-admin-principal` subcommand multiple times.

You can use the "`list-secure-admin-principals`" subcommand to list the certificates for which Payara Server accepts admin requests from clients.

Guarding Against Unwanted Connections

Secure admin guards against unwanted connections in several ways:

- **DAS-to-DAS, instance-to-instance**:
  - The DAS and the instances have copies of the same truststore, which contains the public certificate of the DAS and the separate public certificate that is used by all instances. In addition, Payara Server includes a unique, generated "domain ID" that servers use to ensure that admin requests from other Payara Servers originate from the correct domain.
  - DAS-to-other-DAS communication is not authenticated because each different DAS will have its own self-signed certificate that is not in the truststore of the other DAS.
  - DAS-to-itself communication is unlikely unless you were to mis-configure the admin listener port for an instance on the same host, so it is the same as for the DAS. Similarly, instance-to-instance traffic is unlikely unless you were to mis-configure listener ports for instances on the same host.

    To prevent both of these situations, both cases are handled by making sure that the connecting `Principal` (alias) is not the running `Principal`. Secure admin ensures that if the client has authenticated using SSL/TLS client authentication

that the `Principal` associated with the remote client is not the same as the current process. That is, the DAS makes sure that the `Principal` is not itself. Similarly, each instance ensures that the client is not an instance. (The instances share the same self-signed certificate and therefore are mapped to the same `Principal`)

- Remote client-to-instance:

  Remote `asadmin` clients are unable to connect directly to instances. If the user on host "test1" runs a local command but specifies a remote instance on host "test2," `asadmin` on "test1" will read and send that locally-provisioned password. The instance on "test2" will have a different locally-provisioned password and so the authentication attempt will fail.

  Therefore, a user on "test1" will not be able to run a remote command targeting an instance on "test2."

Considerations When Running Payara Server with Default Security

In Payara Server, the default admin account has the username "admin" with an empty password credentials. By default, admin clients provide empty credentials or none at all, and all are authenticated and authorized as that default admin user. None of the participants (clients, DAS, or instances) encrypts network messages.

If this level of security is acceptable in your environment, no changes are needed, you do not need to enable secure administration.

NOTE	Imposing a heightened level of security is optional, but recommended on most public-facing production environments.

However, consider the following table, which shows which operations are accepted and rejected when secure admin is disabled.

NOTE	When secure admin is disabled, Payara Server does allow remote monitoring (read-only) access via the REST interface.

Table 162. Accepted and Rejected Operations if Secure Admin is Disabled

Operation	Run Locally	Run Remotely
`start-local-instance`	Functions as expected	Cannot sync with DAS. The instance starts but cannot communicate with the DAS. DAS will not properly recognize the instance and synchronize with it.
Any other `asadmin` subcommand	Functions as expected	Rejected. The user sees the credentials prompt, but even correct entries are rejected.
Commands that use SSH. For example, `create-instance`.	Functions as expected but requires prior SSH configuration.	Functions as expected but requires prior SSH configuration.

Running Secure Admin

This section describes how to run secure admin. The section begins with prerequisites for running secure admin.

Prerequisites for Running Secure Admin

Before running Payara Server with secure admin enabled, you must make sure that:

1. The DAS is initialized and running.
2. If one or more remote instances are installed and initialized, they must not be running.
3. Any administration clients you require are installed.
4. The username and password sent by remote administration clients (`asadmin`, administration console, browsers, and IDEs) must exist in the realm and be in the admin group.
5. The keystore and truststore for the domain exist. (They are created by default when you create the domain)

NOTE	If you are not using the default self-signed certificates, you must add your own valid certificates (including CA root certificates) in the

keystore and truststore, respectively.

6. If you are not using the default self-signed certificates, create two aliases corresponding to certificates in the keystore and truststore: one that the DAS will use for authenticating itself in administration traffic, and one that the instances will use for authenticating itself in administration traffic.

TIP For convenience, you can use the same certificate alias for both DAS access and DAS-to-instance communication.

An Alternate Approach: Using a Username and Password for Internal Authentication and Authorization

By default, secure admin uses the Payara Server self-signed certificates, via the aliases corresponding to these certificates, to authenticate the DAS and instances with each other and to authorize secure admin operations. Specifically, the DAS uses the (`s1as`) alias for authenticating itself and authorizing access in administration traffic, and instances use the (`glassfish-instance`) alias for authenticating themselves and authorizing access in secure admin traffic.

As described in using-your-own-certificates, you can instead use your own certificates and their associated aliases for authenticating and authorizing the DAS and instances in administration traffic and communication.

As an alternative to this certificate-based authentication and authorization, you can instead use the enable-secure-admin-internal-user subcommand to instruct all servers in the domain to authenticate to each other, and to authorize admin operations submitted to each other, using an existing admin username and password rather than certificates.

NOTE If secure admin is enabled, all Payara Server processes continue to use SSL encryption to secure the content of the admin messages, regardless of how they authenticate to each other.

You might want to use the enable-secure-admin-internal-user subcommand if your use case favors the use of a username and password combination over the use of SSL certificates and aliases.

This generally means that you must:

1. Create a valid admin user.

```
asadmin> create-file-user --authrealmname admin-realm --groups
asadmin newAdminUsername
```

2. Create a password alias for the password credential of the newly created user.

```
asadmin> create-password-alias passwordAliasName
```

The following example allows secure admin to use a username and password alias for authentication and authorization between the DAS and instances, instead of certificates.

```
asadmin> enable-secure-admin-internal-user --passwordalias passwordAliasName
newAdminUsername
```

If Payara Server finds at least one secure admin internal user, then if secure admin is enabled Payara Server processes will not use SSL authentication and authorization with each other and will instead use basic authentication through username and password credentials.

Most users who use this subcommand will need to set up only one secure admin internal user. If you set up more than one secure admin internal user, you should not make any assumptions about which username and password pair Payara Server will choose to use for any given admin request.

IMPORTANT	As a general practice, you should not use the same username and password pair for internal admin communication and for admin user login. That is, always create at least one admin account

> specifically for internal admin communication if you prefer to use this method.

You can use the disable-secure-admin-internal-user subcommand to disable secure admin from using the username/password credentials to authenticate the DAS and instances with each other and to authorize admin operations.

To disable multiple usernames for authenticating and authorizing access in secure admin, run the `disable-secure-admin-internal-user` subcommand multiple times.

You can use the list-secure-admin-internal-users subcommand to list the usernames for which Payara Server authenticate the DAS and instances with each other and authorizes admin operations.

Additional Considerations When Creating Local Instances

If you use `*-local-instance` variant commands to set up local instances (which are run locally from within the node host where the instance has been installed), either leave secure admin disabled, or enable it before you create or start the instances and leave it that way.

However, if you use `*-instance` variant commands over SSH to manage remote instances (which are run from the DAS), you can enable and disable secure admin, although this is not recommended because it can result in an inconsistent security model.

Secure Admin Use Case

This section describes a simple secure admin use case.

In the `asadmin --secure=false --user me --passwordfile myFile.txt cmd ...` use case, the user submits a command with `--secure` set to false, and supplies password credentials.

The important concept to note is that `asadmin` uses HTTPS because of the DAS redirection, even though the command sets `--secure` to false. `asadmin` sends the

HTTP Authorization header along with the redirected request.

In addition to the flow described here, certificate authentication is also performed as described in the following table. Also, the credentials that the user supplies are assumed to be valid administrator credentials for the DAS.

Table 163. asadmin --secure=false, With Username and Password

`asadmin`	HTTP Engine	AdminAdapter
Sends HTTP request, no authorization header (because the transport is not secure).	+	+
+	Returns `3xx` status and redirects HTTP to HTTPS.	+
Follows redirection, this time adding the Authorization header (because transport is now HTTPS).	+	+
+	+	Authenticates admin user and password from HTTP Authorization header in the realm Executes command, and responds with success status.

Running in a Secure Environment

This chapter describes important information about running Payara Server in a secure environment.

This chapter assumes that you are familiar with security features such as authentication, authorization and certificates. If you are not, see administering-system-security first.

Determining Your Security Needs

Before you deploy Payara Server and your Jakarta EE applications into a production environment, determine your security needs and make sure that you take the

appropriate security measures.

Understand Your Environment

To better understand your security needs, ask yourself the following questions:

Which resources am I protecting?:

Many resources in the production environment can be protected, including information in databases accessed by Payara Server and the availability, performance, applications, and the integrity of the website.

+ Consider the resources you want to protect when deciding the level of security you must provide.

From whom am I protecting the resources?:

For most websites, resources must be protected from everyone on the Internet. But should the website be protected from the employees on an intranet in your enterprise infrastructure?

+ Should your employees have access to all resources within the server environment?

+ Should the system administrators have access to all Payara Server resources?

+ Should the system administrators be able to access all data?

+ You might consider giving access to highly confidential data or strategic resources to only a few well trusted system administrators. Perhaps it would be best to disallow system administrators access to the data or resources.

What will happen if the protections on strategic resources fail?:

In some cases, a fault in your security scheme is easily detected and considered nothing more than an inconvenience. In other cases, a fault might cause great damage to companies or individual clients that use the website.

+ Understanding the security ramifications of each resource will help you protect it

properly.

Installing Payara Server in a Secure Environment

This section describes recommendations for installing Payara Server in a secure environment.

Enable the Secure Administration Feature

The secure administration feature allows an administrator to secure all administrative communication between the domain administration server (DAS), any remote instances, and administration clients such as the asadmin utility, the administration console, and REST clients.

In addition, secure administration helps to prevent DAS-to-DAS and instance-to-instance traffic, and carefully restricts administration-client-to-instance traffic.

When you install Payara Server or create a new domain, secure admin is disabled by default. Payara Server does not encrypt administrative communication among the system components and does not accept administrative connections from remote hosts.

NOTE	Imposing a heightened level of security is optional.

See managing-administrative-security for information on enabling the secure administration feature.

Run on the Web Profile If Possible

If your applications can run without issues on Payara Server's Web Profile, use that instead of the Full Profile. As the web profile is a subset of the full feature set of Azul Payara, it has fewer features that require extra considerations for security (like ORB listeners, Web Services, Message Queue Brokers) which reduces the overall risks of the server's installation.

Securing the Payara Server Host Machine

A Payara Server production environment is only as secure as the security of the

machine on which it is running. It is important that you secure the physical machine, the operating system, and all other software that is installed on the host machine.

The following are basic recommendations for securing a Payara Server host in a production environment. Also check with the manufacturer of the machine and operating system for recommended security measures.

IMPORTANT | The domain and server configuration files should be accessible only by the operating system users who configure or execute Payara Server processes.

Table 164. Securing the Payara Server Host Machine

Security Action	Description
Physically secure the hardware.	Keep your hardware in a secured area to prevent unauthorized operating system users from tampering with the deployment machine or its network connections.
Log out of the Administration Console before navigating to a non-secure site.	If you are logged on to the Administration Console, be sure to log out completely before browsing to an unknown or non-secure website.
Secure networking services that the operating system provides.	Have a security professional review network services such as e-mail programs or directory services to ensure that a malicious attacker cannot access the operating system or system-level commands. These actions depend on the operating system used by the host machine. Sharing a file system with other machines in the enterprise network imposes risks of a remote attack on the file system. Be certain that the remote machines and the network are secure before sharing the file systems from the machine.
Use a file system that can prevent unauthorized access.	Make sure that the file system on each Payara Server host can prevent unauthorized access to protected resources. **TIP** | For example, on a Windows host, use only exFAT.

Security Action	Description
Set file access permissions for data stored on disk.	Set operating system file access permissions to restrict access to data stored on disk. This data includes, but is not limited to, the following: • The database files. Payara Server includes an integrated H2 database,however, you can use any JDBC-compliant database. • The directory and filename location of a private keystore, such as `keystore.p12` • The directory and filename location of a Root Certificate Authority (CA) truststore, such as `cacerts.p12`. • Nix operating systems provide utilities such as `umask` and `chmod` to set the file access permissions. At a minimum, consider using `umask 066`, which denies read and write permission to the group owner and other users.
Set file access permission for the Payara Server installation.	The directory structure in which Payara Server is located, including all files, should be protected from access by unprivileged users. Taking this step helps ensure that unprivileged users cannot insert code that can potentially be executed by Payara Server.

Security Action	Description
Limit the number of user accounts on the host machine.	Avoid creating more user accounts than you need on host machines, and limit the file access privileges granted to each account. On operating systems that allow more than one system administrator user, the host machine should have two user accounts with system administrator privileges and one user with sufficient privileges to run Payara Server. Having two system administrator users provides a backup at all times. The Payara Server user should be a restricted user, not a system administrator user. One of the system administrator users can always create a new Payara Server user if needed. **IMPORTANT** Domain and server configuration files should be accessible only by the operating system users who configure or execute Payara Server. **CAUTION** Review active user accounts regularly and according to your organisation account management policies. Configuration data and some URL (Web) resources, including Java Server Pages (JSPs) and HTML pages, are stored in clear text on the file system. A sophisticated user or intruder with read access to files and directories might be able to defeat any security mechanisms you establish with authentication and authorization schemes.
For your system administrator user accounts, choose names that are not obvious.	For additional security, avoid choosing an obvious name such as "system," "admin," or "administrator" for your system administrator user accounts.

Security Action	Description
Safeguard passwords.	The passwords for user accounts on production machines should be robust, difficult to guess and should be guarded carefully. Set a policy to expire passwords periodically based on your organisation account management policies. **WARNING** | Never code passwords in client applications.
Safeguard password files	The `-passwordfile` option of the `asadmin` command specifies the name of a file that contains password entries in a specific format. These password entries are stored in clear text in the password file, and rely on file system mechanisms for protection. Therefore, any password file created for use with the `-passwordfile` option should be protected with file system permissions. Additionally, any password file being used for a transient purpose, such as setting up SSH among nodes, should be deleted after it has served its purpose.
Use password aliases	A password alias stores a password in encrypted form in the domain keystore, providing a clear-text alias name to use instead of the password. To provide additional security, use the `create-password-alias` subcommand to create an alias for the password. The password for which the alias is created is stored in an encrypted form.
Do not run Payara Server as root	Payara Server should run only as an unprivileged user, never as the `root` in *Nix systems or as an Administrator user in Windows systems.
Do not develop code on a production machine.	Develop your applications first on a development machine and then move code to the production machine when it is completed and tested. This process prevents bugs in the development environment from affecting the security of the production environment.
Do not install development or sample software on a production machine.	Do not install development tools on production machines. Keeping development tools off the production machine reduces the leverage intruders have should they get partial access to a production machine.

Security Action	Description
Enable security auditing.	If the operating system on which Payara Server runs supports security auditing of file and directory access, it is recommended using audit logging to track any denied directory or file access violations. **NOTE** Administrators should ensure that sufficient disk space is available for the audit log.
Consider using additional software to secure your operating system.	Most operating systems can run additional software to secure a production environment. For example, an Intrusion Detection System (IDS) can detect attempts to modify the production environment. Refer to the vendor of your operating system for information about available software that can be used for this purpose.
Apply operating system patch sets and security patches.	Refer to the vendor of your operating system for a list of recommended patch sets and security-related patches.
Apply Java runtime security patches.	Refer to your official Java runtime version for a list of recommended security-related patches. **NOTE** In general, whenever possible it is recommended to upgrade the Java runtime to its latest security patch on its corresponding major version.

Securing Payara Server

Payara Server provides a powerful and flexible set of software tools for securing the subsystems and applications that run on a server instance. The following table provides a checklist of essential features recommended to secure your production environment.

Table 165. Securing Payara Server

Security Action	Description
Enable Secure Admin.	The secure administration feature allows an administrator to secure all administrative communication between the domain administration server (DAS), any remote instances, and administration clients such as the `asadmin` utility, the administration console, and REST clients. In addition, secure administration helps to prevent DAS-to-DAS and instance-to-instance traffic, and carefully restricts administration-client-to-instance traffic. The secure administration feature provides a secure environment, in which you can be confident that rogue users or processes cannot intercept or corrupt administration traffic or impersonate legitimate Payara Server components. See managing-administrative-security.
Protect the `.asadminpass` file	If you create a domain with the `--savelogin` option, `create-domain` saves the administration username and password in the `.asadminpass` file in the user's home directory. Make sure that this file remains protected. Information stored in this file will be used by `asadmin` commands to manage the domain.
Safeguard password files	The `-passwordfile` option of the `asadmin` command specifies the name of a file that contains password entries in a specific format. These password entries are stored in clear text in the password file, and rely on file system mechanisms for protection. Therefore, any password file created for use with the `-passwordfile` option should be protected with file system permissions. Additionally, any password file being used for a transient purpose, such as setting up SSH among nodes, should be deleted after it has served its purpose. **NOTE** | To provide additional security, create a password alias.

Security Action	Description
Use SSL/TLS secured transport, but do not use the self-signed certificates in a production environment.	To prevent sensitive data from being compromised, secure data transfers by using HTTPS. By default, Payara Server uses self-signed certificates. The self-signed certificates that Payara Server uses will not be trusted by clients by default because a certificate authority does not vouch for their authenticity. You can instead use your own custom CA-issued certificates, as described in using-your-own-certificates.
Restrict the size and the time limit of requests on external channels to prevent **Denial of Service (DoS) attacks**.	To prevent some Denial of Service (DoS) attacks, restrict the size of a message as well as the maximum time it takes a message to arrive. **NOTE** The default setting for maximum post size is `2097152` bytes and `900` seconds for the request timeout.
Set appropriate logging for security and SSL log entries.	Consider setting module log levels for the `javax.enterprise.system.ssl.security.` and `javax.enterprise.system.core.security.` You can set a level from `SEVERE` to `FINEST` (the default is `INFO`), but be aware that the finer logging levels may produce a large log file. By default, Payara Server logging messages are recorded in the server log, and you can set the file rotation limit, as described in "`rotate-log`"
Ensure that you have correctly assigned users to the correct groups.	Make sure you have assigned the desired set of users to the right groups. In particular, make sure that users assigned to the `asadmin` group need to be members of that group.
Create no fewer than two user accounts in the `asadmin` group.	The default `admin` user is created when you install Payara Server or create a new domain and accept its default username. For production environments, create at least one other account in the `asadmin` group in case one account password is compromised. When creating asadmin users give them unique names that cannot be easily guessed.

Securing Applications

Although much of the responsibility for securing Payara Server resources in a domain fall within the scope of the server, some security responsibilities lie within the scope of individual applications.

For some security options, Payara Server enables you to determine whether the server or individual applications are responsible. For each application that you deploy in a production environment, review the items in the following table to verify that you have properly secured its resources.

Table 166. Securing Applications

Security Action	Description
Use JSP comment tags instead of HTML comment tags.	Comments in JSP files that might contain sensitive data and or other comments that are not intended for the end user should use the JSP syntax of `<%/* xxx */%>` instead of the HTML syntax `<!-- xxx -¬`. JSP comments, unlike HTML comment blocks, are deleted when the JSP is compiled and therefore cannot be viewed in a web browser.
Do not install un-compiled JSPs and other source code on a production machine.	Always keep source code off of the production machine. Getting access to your source code allows an intruder to find security holes. Consider pre-compiling JSPs and installing only the compiled JSPs on the production machine. To do this, set the `deploy` subcommand `-precompilejsp` option to true for the component. When set to true, the `deploy` and `redeploy` subcommands `-precompilejsp` option compiles JSPs during deploy time. If set to `false` (the default), JSPs are compiled during runtime.
Configure your applications to use secure transport.	Set the `transport-guarantee` to `CONFIDENTIAL` in the `user-data-constraint` element of the `web.xml` deployment descriptor whenever appropriate.

Security Action	Description
Examine applications for security.	There are instances where an application's code can lead to a security vulnerability. Of particular concern is code that uses Java native interface (JNI) because Java positions native code outside the scope of the Java security model. If Java native code behaves in an errant manner, it is only constrained by the operating system. That is, the Java native code can do anything Payara Server itself can do. This potential vulnerability is further complicated by the fact that buffer overflow errors are common in native code and can be used to run arbitrary code.
If your applications contain untrusted code, enable the Java security manager.	The Java security manager defines and enforces permissions for classes that run within a JVM. In many cases, where the threat model does not include malicious code being run in the JVM, the Java security manager is unnecessary. However, when third parties use Payara Server and untrusted classes are being run, the Java security manager may be useful. See "enabling-and-disabling-the-security-manager" in the Application Development section.
Replace HTML special characters when servlets or JSPs return user-supplied data.	The ability to return user-supplied data can present a security vulnerability called cross-site scripting (XSS), which can be exploited to steal a user's security authorization. To remove this security vulnerability, before you return data that a user has supplied, scan the data for HTML special characters. If you find any such characters, replace them with their HTML entity or character reference. Replacing the characters prevents the browser from executing the user-supplied data as HTML. **NOTE** | Using third-party libraries that can sanitize the content of an HTML-formatted set of data is recommended in most cases.

SSL Certificate Management

As a reminder, Payara Server uses the following PKCS12 keystore files to manage SSL/TLS certificates required for secure communications:

`keystore.p12`: Java Keystore formatted file containing the Cryptographic keys, like the private keys associated with the certificates. `cacerts.p12`: Java Keystore formatted file used as Trust Store file holding the X.509 certificates.

These keystores are protected with the Master Password of your domain.

Adding a New SSL Certificate

To add your SSL Certificate to the Payara Server configuration files, you need:

- The private key associated with your certificate.
- The Certificate you received from the Certificate Authority based on your Certificate Signing Request (CSR).

You can generate a self-signed certificate using the "`generate-self-signed-certificate`" command:

```
asadmin generate-self-signed-certificate --dn "CN=test.payara.fish,IP=127.0.0.1"
mydomain_certificate
```

Alternatively: You can find the detailed steps of adding a self-signed certificate into Payara Server for testing purposes using `keytool` and `openssl` here.

The private key can be stored in various formats. We assume that the key is in a format which can be read by the Java key tool program, such as PKCS12.

First, you'll have to combine the Private key and the Certificate you generated into a single file. This file can then be used to merge our data into the files used by the server.

Then specify an alias name for your key/certificate. This alias name is important as it will be used to refer to the SSL Certificate as a complete unit.

The command to perform this combination is the following with OpenSSL:

```
openssl pkcs12 -export -in mycert.crt -inkey mycert.key -out mycert.p12 -name
mydomain_certificate
```

The resulting P12 bundle file can then be added to the keystore and truststore in Payara Server as described in the following sections.

Import into the Keystore

The information needs to be imported into the `keystore.p12` file of your Payara Server domain which will use it.

Using Asadmin CLI

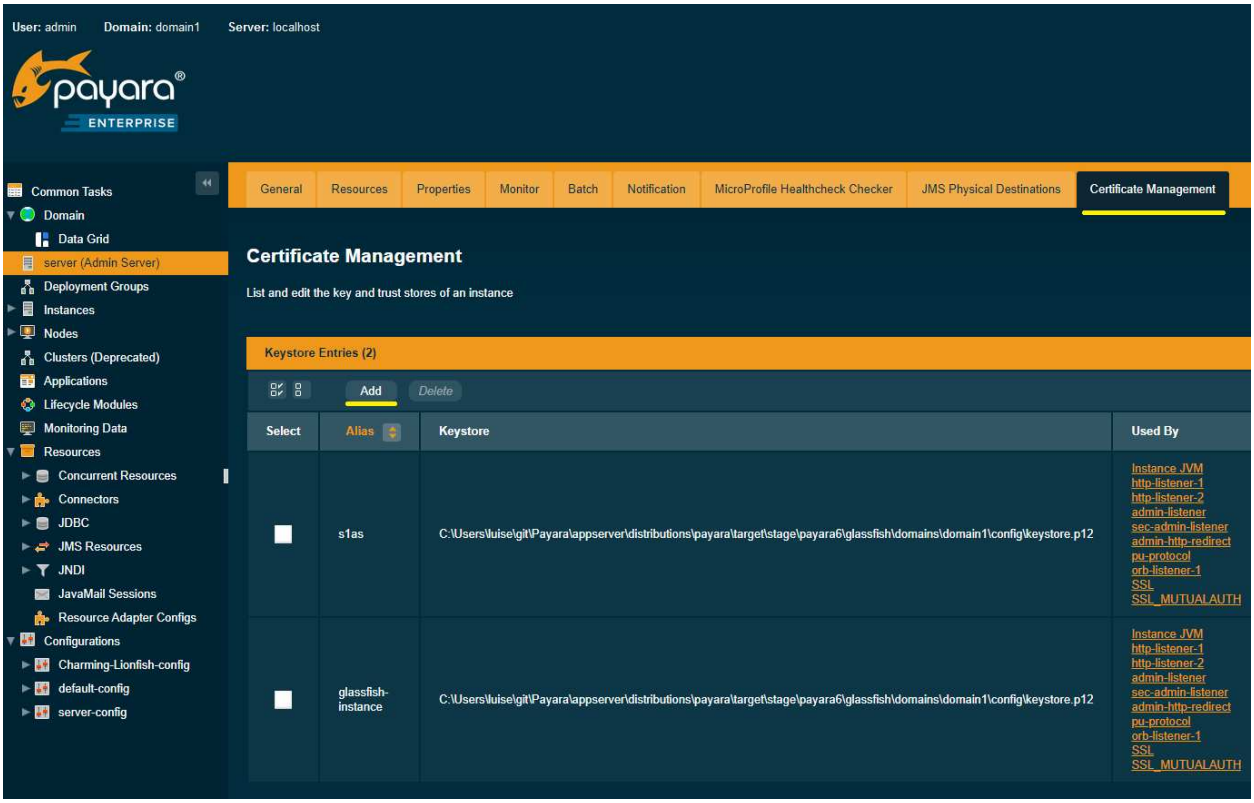
You can use the "`add-to-keystore`" command:

```
asadmin add-to-keystore --file mycert.p12 mydomain_certificate
```

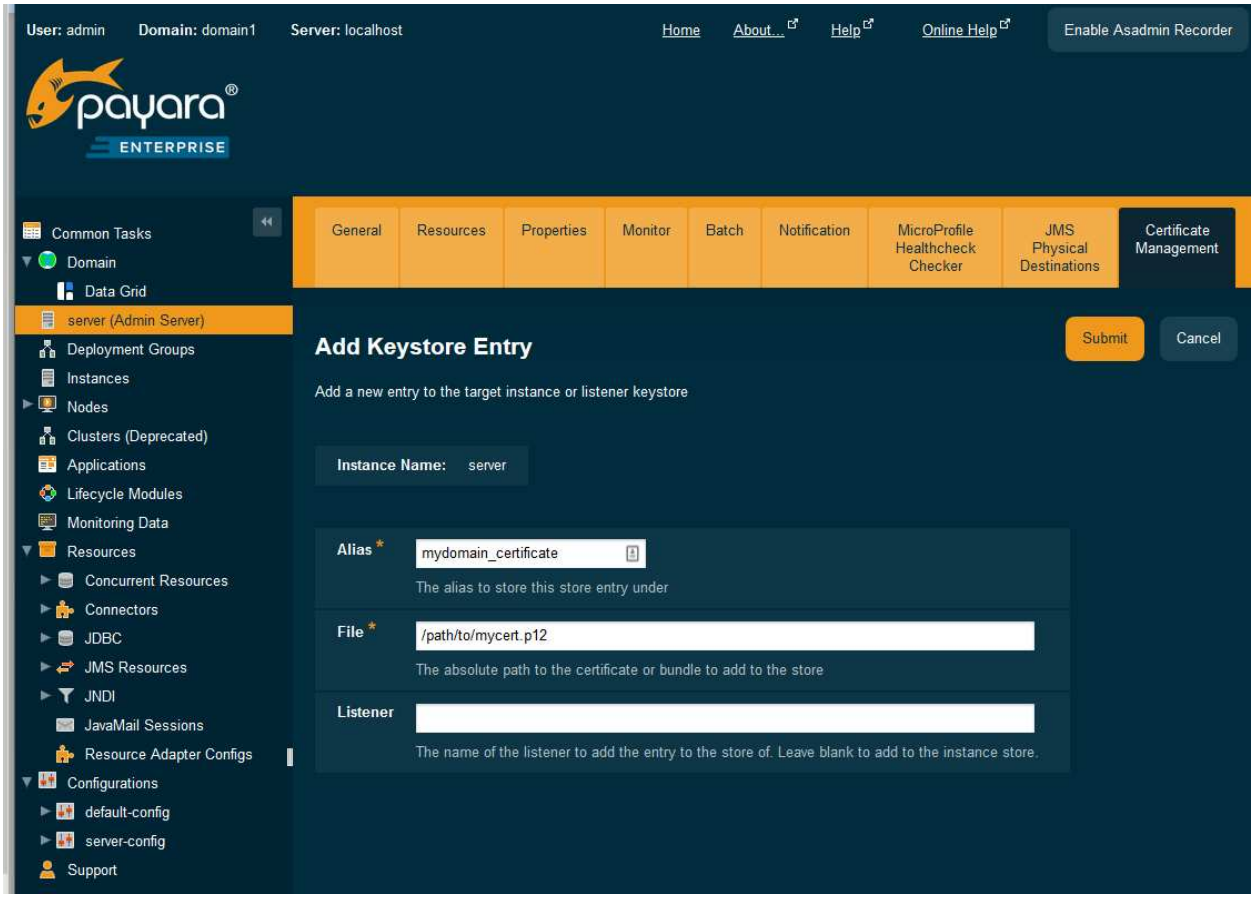
Using the Admin Console

From the Admin Console, open the instance details page (e.g. on the navigation bar select *Instances* → *$instanceName*) and navigate to the *Certificate Management* tab.

From here, Click on the *Add* button of the keystore table (highlighted):



You can then enter the alias name and the path to the certificate bundle:



Using the `keytool` Command

You can also use the `keytool` command of the JDK:

```
keytool -importkeystore -destkeystore keystore.p12 -srckeystore mycert.p12
-srcstoretype PKCS12 -alias mydomain_certificate
```

Import into the Truststore

A similar command is used to import the information into the Trust Store file so that our certificate is trusted by the JVM.

Using Asadmin CLI

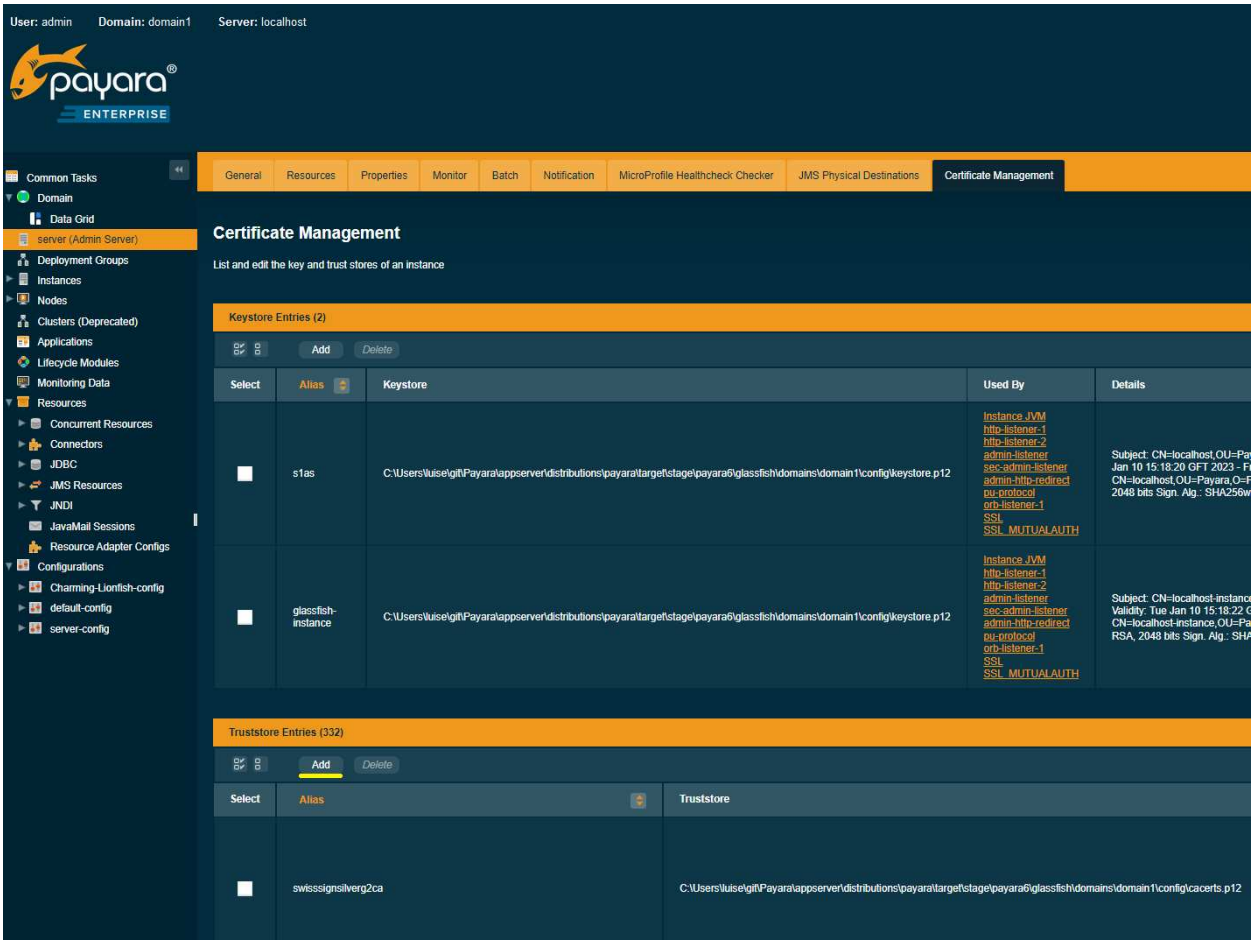
You can use the "`add-to-truststore`" command:

```
asadmin add-to-truststore --file mycert.crt mydomain_certificate
```

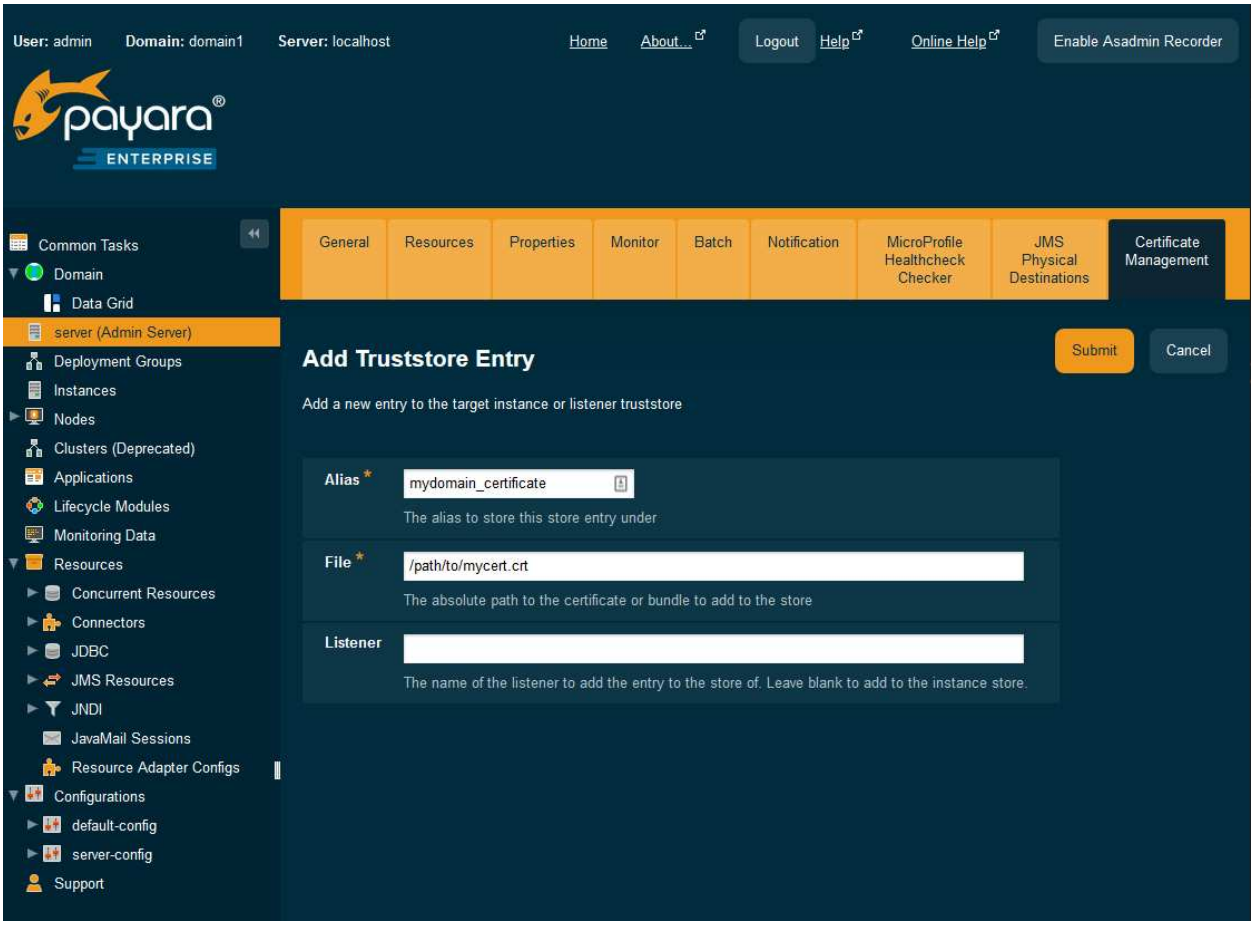
Using the Admin Console

You can also follow the same steps as done for adding the bundle to the keystore via

the admin console, though selecting the **Add** button of the truststore table



And then specifying the `mycert.crt` file.



Using the `keytool` Command

You can also use the `keytool` command of the JDK:

```
keytool -importcert -trustcacerts -destkeystore cacerts.p12 -file mycert.crt -alias
mydomain_certificate
```

Update HTTP Listeners

The last step you have to perform is to configure which certificate needs to be used when the server receives an SSL/TLS request. This can be done by specifying the alias name we have used when we integrated the required information into the key store files of the domain.

Using Asadmin CLI

This information can be set via the CLI:

```
asadmin set configs.config.server-config.network-config.protocols.protocol.http-
```

```
listener-2.ssl.cert-nickname=mydomain_certificate
```

NOTE

In a default installation, the `http-listener-2` listener is responsible for handling the secure connections on port **8181**. The above command sets the alias name so that the system knows which certificate needs to be used.

Using the Admin Console

The information can also be entered in the Web Administration console, by navigating to the page Configurations `server-config` HTTP Service HTTP Listeners `http-listener-2`.

Loading Certificates from Multiple Keystores

IMPORTANT

All additional keystores must have the same password as your domain master password.

To add additional keystores to Payara Server you will need to have your keystore and truststore files in a local directory. You can then configure this using the admin console or asadmin commands. If you want to load more than one additional keystore, you will need to use a delimiter between the paths to the additional keystores. This is different depending on your OS, Windows uses ';' and Linux uses ':'.

NOTE

When specifying the key or trust store for a specific listener, this is deemed as an absolute and any additional key or trust stores are not considered.

The new JVM properties used to add additional keystores are:

```
-Dfish.payara.ssl.additionalKeyStores
-Dfish.payara.ssl.additionalTrustStores
```

Using the Admin Console

To configure the additional keystore locations in the admin console, head to the **Configurations** **<instance configuration>** **JVM Settings** and on the **JVM Options** tab

click **Add JVM Option**.

You can add the new JVM property and the relative paths to your keystores or truststores here.

[Configure Additional keystores and truststores in Admin Console]

Using Asadmin CLI

JVM options can be configured using the `create-jvm-options` asadmin command, you can configure your additional keystore and truststore files using this command. If you are loading in multiple additional keystores via asadmin commands, you will need to prefix the appropriate delimiter for your OS with `\` to avoid creating multiple JVM options.

Additional Keystores

```
asadmin create-jvm-options "-
Dfish.payara.ssl.additionalKeyStores=/path/to/keystore.p12\:/path2/to/keystore2.p12
"
```

Additional Truststores

```
asadmin create-jvm-options "-
Dfish.payara.ssl.additionalTrustStores=/path/to/truststore.p12\:/path2/to/truststor
e2.p12"
```

NOTE	If you load multiple keystores with the same alias, the server will use the first keystore with that alias, starting with the default and then the additional keystores in the order they are listed in the JVM option.

Certificate Expiration

All X.509 certificates have a validity period when they can be used. Once this validity period is passed, the users will see a warning or error message depending on the browser that the certificate is no longer valid.

Within the server log file, the expired certificates are listed when the system encounters

one. Besides your custom certificates which are added as described in a previous chapter, the Trust Store also contains certificates from recognized Certificate Authorities, which can also expire in due time and thus can be listed as entries in the log file too.

NOTE	The log level of the expired certificates is of type **WARNING**.

Removing Expired Certificates

Using Asadmin CLI

If you wish to remove all expired certificates, you can use the following Asadmin CLI subcommands:

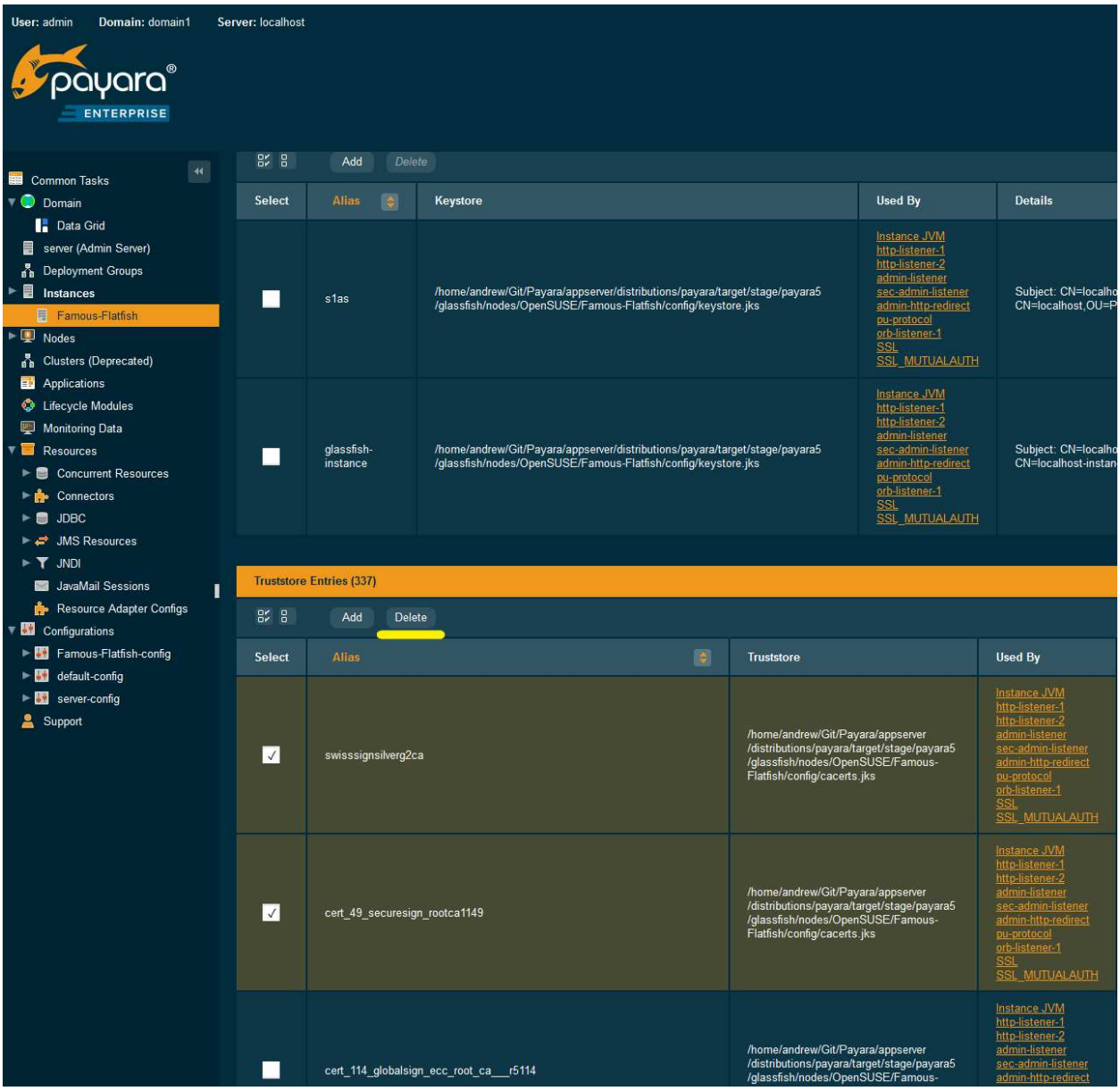
- remove-expired-certificates,
- remove-from-keystore, and
- remove-from-truststore respectively.

```
asadmin remove-expired-certificates
asadmin remove-from-keystore mydomain_certificate
asadmin remove-from-truststore mydomain_certificate
```

Using the Admin Console

You can also remove individual or groups of certificates using the admin console Certificate Management tab (*Instances* *$instanceName* *Certificate Management*).

Select the desired certificates from the keystore **or** truststore entries table (not both), and click on the *Delete* button:



Printing Certificate Data

The `print-certificate` asadmin subcommand is useful to print legible and detailed information of one or multiple SSL certificate and key entries.

Since the output generated by the standard `keytool` utility is not standardized (especially distinguished names since they usually they use RFC 1779), it is recommended to use this command instead.

This command prints the certificate's distinguished name in RFC 2253 format, which is used by Azul Payara, so it can be directly used for principal and role mappings when

using client certificate authentication, for example.

The command supports only X.509 certificates files with the following extensions:

- Keystore format: `p12`, `pkcs12`, `pfx`, `jks`, `jceks`
- PEM or DER encoded certificates: `.cer`, `.cert`, `.crt`, `.der`, `.pem`

NOTE	If the file contains more certificates, the command prints only the first one.

Usage

The `print-certificate` is an offline command (it doesn't need a running domain or server instance as target) that prints data of a selected certificate in RFC 2253 format.

The details that are printed out by the command are the following :

1. Validity dates
2. Serial number
3. Version
4. Issuer's distinguished name (DN)
5. Public key's algorithm and size
6. Signature algorithm including OI

The command will not be successful in one of the following cases:

1. The file cannot be read (permission issues or I/O errors)
2. The file extension is not supported by the command
3. The file type is not supported by the JDK
4. The keystore password is not correct
5. The keystore does not contain a certificate entry under the given alias
6. The certificate type is not supported

See the "`print-certificate`" documentation in the Command Reference for more information on its structure.

Examples

DER/PEM Encoded Certificate

Here's an example for printing a DER certificate with one entry:

```
asadmin print-certificate ./certificate.der

Found Certificate:
Subject:    UID=LDAP-Test,EMAILADDRESS=nobody@nowhere.space,CN
=PrintCertificateCommandTest,OU=Test Test\, Test,O=Payara Foundation,L=Pilsen,C=CZ
Validity:   Thu Aug 01 02:00:00 CEST 2019 - Fri Aug 02 02:00:00 CEST 2019
S/N:        1
Version:    3
Issuer:     UID=LDAP-Test,EMAILADDRESS=nobody@nowhere.space,CN
=PrintCertificateCommandTest,OU=Test Test\, Test,O=Payara Foundation,L=Pilsen,C=CZ
Public Key: RSA, 2048 bits
Sign. Alg.: SHA256withRSA (OID: 1.2.840.113549.1.1.11)
Command print-certificate executed successfully.
```

Keystore

Here's an example for printing a specific certificate using a P12 format keystore:

```
asadmin print-certificate --certificatealias s1as ./keystore.p12

Keystore Password>
Found Certificate:
Subject:    CN=localhost,OU=Payara,O=Payara Foundation,L=Great Malvern,ST
=Worcestershire,C=UK
Validity:   Tue Aug 06 14:06:14 CEST 2019 - Fri Aug 03 14:06:14 CEST 2029
S/N:        886895448
Version:    3
Issuer:     CN=localhost,OU=Payara,O=Payara Foundation,L=Great Malvern,ST
=Worcestershire,C=UK
Public Key: RSA, 2048 bits
Sign. Alg.: SHA256withRSA (OID: 1.2.840.113549.1.1.11)
Command print-certificate executed successfully.
```

Alternative Provider

It is possible to plug a different `java.security.Provider` implementation to get different outputs when running the `print-certificate` command.

In this example, we use the **BouncyCastle** third-party security provider, which leads to one significant difference: A highlighted, uppercase signature's algorithm name.

```
#Copy the library to the asadmin lib directory first
cp bcprov-jdk15on-1.62.jar ./payara7/glassfish/lib/asadmin/

#Then run the command
asadmin print-certificate --providerclass
org.bouncycastle.jce.provider.BouncyCastleProvider ./certificate.der

Found Certificate:
Subject:    UID=LDAP-Test,EMAILADDRESS=nobody@nowhere.space,CN
=PrintCertificateCommandTest,OU=Test Test\, Test,O=Payara Foundation,L=Pilsen,C=CZ
Validity:   Thu Aug 01 02:00:00 CEST 2019 - Fri Aug 02 02:00:00 CEST 2019
S/N:        1
Version:    3
Issuer:     UID=LDAP-Test,EMAILADDRESS=nobody@nowhere.space,CN
=PrintCertificateCommandTest,OU=Test Test\, Test,O=Payara Foundation,L=Pilsen,C=CZ
Public Key: RSA, 2048 bits
Sign. Alg.: SHA256WITHRSA (OID: 1.2.840.113549.1.1.11)
Command print-certificate executed successfully.
```

TIP The asadmin command has its own classpath, so first you need to add the provider's library to the asadmin utility's classpath as shown above.

Command Reference

Overview

This section contains reference collection of all Asadmin CLI subcommands available for the Administration interface of Payara Server and a comprehensive detail of what each command does and how it operates.

Here's a brief description of the structure each command page section and the information it references:

Name: This section gives the names of the commands or functions documented, followed by a brief description of what they do.

Synopsis: This section shows the syntax of commands or functions.

+ The following special characters are used in this section:

+ `[ ]`;; Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified. `|`;; Separator. Only one of the arguments separated by this character can be specified at a time.

Description: This section defines the functionality and behavior of the service. Thus, it describes concisely what the command does. It does not discuss options or cite examples . **Options**: This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the Synopsis section.

+ Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

Operands: This section lists the command operands and describes how they affect the actions of the command.

Examples: This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the Synopsis, Description, Options, and Usage sections.

Exit Status: This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero (`0`) is returned for successful completion, and values other than zero for various error conditions.

See Also: This section lists references to other man pages, in-house documentation, and outside publications.

Notes: This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest.

Domain

Domains have their own configurations.

Description

A domain provides a common authentication and administration point for a collection of zero or more server instances. The administration domain encompasses several

manageable resources, including instances, clusters, and their individual resources. A manageable resource, such as a server instance, may belong to only one domain.

See Also

asadmin-1m

Instance

An instance in Payara Server has its own Jakarta EE configuration, Jakarta EE resources, application deployment areas, and server configuration settings.

Description

Payara Server creates one server instance, called `server` at the time of installation. You can delete the server instance and create a new instance with a different name.

For many users, one server instance meets their needs. However, depending upon your environment, you might want to create additional server instances. For example, in a development environment you can use different server instances to test different Payara Server configurations, or to compare and test different application deployments.

Because you can easily add or delete a server instance, you can use them to create temporary "sandbox" areas to experiment with while developing.

Configuration

The data set that determines how a Payara Server instance operates.

Description

The configuration of Payara Server is the data set that determines how it operates. Parts of this configuration determine the operation of specific parts of Payara Server, such as the following:

- Services, such as the transaction service
- Resources, such as databases
- Deployed applications or modules, such as web applications

- Clusters and server instances

The term configuration is also used to describe a part of the overall configuration, such as the transaction service configuration or the configuration of a database. In clustered environments, clusters, deployment groups or server instances can share configurations.

Examples of configuration data are port numbers, flags that enable or disable processes, application names, and so on. Most of these data points are name/value pairs, either hard-coded attributes or more flexibly defined properties.

The hierarchical structure of the configuration is explained in the dotted names page. You can view and change most of the Payara Server configuration using either the Administration Console or the `asadmin` utility and its subcommands.

To list the structure of all or part of the configuration, use the `list` subcommand. To view the value of one or more attributes or properties, use the `get` subcommand. To change the value of an attribute or property, use the `set` subcommand.

See Also

- get,
- list,
- set,
- asadmin-1m,
- dotted-names
- configuration-tasks in the Payara Server General Administration section

Dotted Names

Syntax for using periods to separate name elements.

Description

A dotted name is an identifier for a particular Payara Server element, such as a configurable or a monitorable object. A dotted name uses the period (`.`), known as dot,

as a delimiter to separate the parts of an element name. The period in a dotted name is similar to the slash (`/`) character that delimits the levels in the absolute path name of a file in the UNIX file system.

The subcommands of the `asadmin` utility use dotted names as follows:

- The `list` subcommand provides the fully qualified dotted names of the management components' attributes.
- The `get` subcommand provides access to the attributes.
- The `set` subcommand enables you to modify configurable attributes and set properties.

The configuration hierarchy is loosely based on the domain's schema document, and the attributes are modifiable. The attributes of the monitoring hierarchy are read-only.

The following format is used for configuration dotted names:

```
config-name.config-element-name.primary-key.attribute-name | instance-name.config-element-name.primary-key.attribute-name
```

The following format is used for resource dotted names:

```
server-name.resource-name.primary-key.attribute-name | domain.resources.resource-name.primary-key.attribute-name
```

The following rules apply to forming dotted names:

- The top-level is configuration, server, or domain name. For example, `server-config` (default configuration), `server` (default server), or `domain1` (default domain).
- A dot (.) always separates two sequential parts of the name.
- A part of the name usually identifies a server subsystem or its specific instance. For example, `web-container`, `log-service`, `thread-pool-1`.
- If any part of the name itself contains a dot (`.`), then the dot must be escaped with a

leading `\` (backslash) so that the `.` (dot) does not act like a delimiter. For further information on escape characters, see the asadmin-1m help page.

- An `*` (asterisk) character can be used anywhere in the dotted name and acts like the wildcard character in regular expressions. Additionally, an `\*` can collapse all the parts of the dotted name.

  For example, a long dotted name such as `this.is.really.long.hierarchy` can be abbreviated to `th*.hierarchy`. The `.` (dot) always delimits the parts of the dotted name.

  NOTE

  Characters that have special meaning to the shell or command interpreter, such as `\*` (asterisk), should be quoted or escaped as appropriate to the shell, for example, by enclosing the argument in quotes. In multimode, quotes are needed only for arguments that include spaces, quotes, or backslash.

- The `--monitor` option of the `get` and `list` subcommands selects the monitoring or configuration hierarchy. If the subcommand specifies `--monitor=false` (the default), the configuration hierarchy is selected. If the subcommand specifies `--monitor=true`, the monitoring hierarchy is selected.
- If you know the complete dotted name and do not need to use a wildcard, the `list`, `get`, and `set` subcommands treat the name differently:
  - The `list` subcommand treats a complete dotted name as the name of a parent node in the abstract hierarchy. When you specify this name to the `list` subcommand, the names of the immediate children at that level are returned. For example, the following command lists all the web modules deployed to the domain or the default server:

    ```
    asadmin> list server.applications.web-module
    ```

  - The`get` and `set` subcommands treat a complete dotted name as the fully qualified name of the attribute of a node (whose dotted name itself is the name

that you get when you remove the last part of this dotted name). When you specify this name to the `get` or `set` subcommand, the subcommand acts on the value of that attribute, if such an attribute exists. You will never start with this case because in order to find out the names of attributes of a particular node in the hierarchy, you must use the * wildcard character.

For example, the following dotted name returns the context root of the web application deployed to the domain or default server:

```
server.applications.web-module.JSPWiki.context-root
```

Examples

Example 1 Listing All Configurable Elements

This example lists all the configurable elements.

```
asadmin> list *
```

Output similar to the following is displayed:

```
applications
configs
configs.config.server-config
configs.config.server-config.admin-service
configs.config.server-config.admin-service.das-config
configs.config.server-config.admin-service.jmx-connector.system
configs.config.server-config.admin-service.property.adminConsoleContextRoot
configs.config.server-config.admin-service.property.adminConsoleDownloadLocation
configs.config.server-config.admin-service.property.ipsRoot
configs.config.server-config.ejb-container
configs.config.server-config.ejb-container.ejb-timer-service
configs.config.server-config.http-service
configs.config.server-config.http-service.access-log
configs.config.server-config.http-service.virtual-server.__asadmin
configs.config.server-config.http-service.virtual-server.server
configs.config.server-config.iiop-service
configs.config.server-config.iiop-service.iiop-listener.SSL
configs.config.server-config.iiop-service.iiop-listener.SSL.ssl
configs.config.server-config.iiop-service.iiop-listener.SSL_MUTUALAUTH
configs.config.server-config.iiop-service.iiop-listener.SSL_MUTUALAUTH.ssl
configs.config.server-config.iiop-service.iiop-listener.orb-listener-1
configs.config.server-config.iiop-service.orb
configs.config.server-config.java-config
configs.config.server-config.jms-service
configs.config.server-config.jms-service.jms-host.default_JMS_host
configs.config.server-config.mdb-container
```

```
configs.config.server-config.monitoring-service
configs.config.server-config.monitoring-service.module-monitoring-levels
...
property.administrative.domain.name
resources
resources.jdbc-connection-pool.DerbyPool
resources.jdbc-connection-pool.DerbyPool.property.DatabaseName
resources.jdbc-connection-pool.DerbyPool.property.Password
resources.jdbc-connection-pool.DerbyPool.property.PortNumber
resources.jdbc-connection-pool.DerbyPool.property.User
resources.jdbc-connection-pool.DerbyPool.property.connectionAttributes
resources.jdbc-connection-pool.DerbyPool.property.serverName
resources.jdbc-connection-pool.__TimerPool
resources.jdbc-connection-pool.__TimerPool.property.connectionAttributes
resources.jdbc-connection-pool.__TimerPool.property.databaseName
resources.jdbc-resource.jdbc/__TimerPool
resources.jdbc-resource.jdbc/__default
servers
servers.server.server
servers.server.server.resource-ref.jdbc/__TimerPool
servers.server.server.resource-ref.jdbc/__default
system-applications
Command list executed successfully.
```

Example 2 Listing All the Monitorable Objects

The following example lists all the monitorable objects.

```
asadmin> list --monitor *
```

Output similar to the following is displayed:

```
server
server.jvm
server.jvm.class-loading-system
server.jvm.compilation-system
server.jvm.garbage-collectors
server.jvm.garbage-collectors.Copy
server.jvm.garbage-collectors.MarkSweepCompact
server.jvm.memory
server.jvm.operating-system
server.jvm.runtime
server.network
server.network.admin-listener
server.network.admin-listener.connections
server.network.admin-listener.file-cache
server.network.admin-listener.keep-alive
server.network.admin-listener.thread-pool
server.network.http-listener-1
server.network.http-listener-1.connections
server.network.http-listener-1.file-cache
server.network.http-listener-1.keep-alive
server.network.http-listener-1.thread-pool
server.transaction-service
```

```
Command list executed successfully.
```

See Also

- asadmin-1m
- get,
- list,
- set

Deployment Group

A deployment group is a loose grouping of Payara Server instances that can be used as a target for **deployments**. Deployment Groups consist of one or more standalone instances and each instance can belong to one or more deployment groups.

Description

Deployment groups are effective and more flexible replacements for traditional clusters, as they can be used to effectively manage the lifecycle of multiple server instances at once.

When you create a Deployment Group you can add and remove existing standalone instances from the group. However, Deployment Groups provide greater flexibility than traditional clusters:

- An instance can be in many different deployment groups.
- You can add an instance to a deployment group at any time and any targeted applications or resources will automatically be deployed.
- You can also remove an instance from a deployment group and applications and resources will be un-deployed.

Payara Server instances in the deployment group are not hidden in the administration console and are still visible for management under the 'Instances' section of the admin console navigation tree.

Applications

Server-side Jakarta EE applications and web services.

Description

The Jakarta EE platform enables applications to access systems that are outside of the application server. Applications connect to these systems through resources.

The Payara Server infrastructure supports the deployment of many types of distributed applications and is an ideal foundation for building applications based on Service Oriented Architectures (SOA). SOA is a design methodology aimed at maximizing the reuse of application services. These features enable you to run scalable and highly available Jakarta EE applications.

See Also

- create-application-ref,
- deploy,
- list-applications

Auto-Naming

Auto naming is an `asadmin` feature added to help with the naming of instances, deployment groups, and other resources by resolving name conflicts and auto-generating names for dynamic resources.

Usage

The `autoname` option is enabled either by specifying `--autoname` or `-a` on the command line before or after the command name. This is technically an `asadmin` option rather than a command parameter so should be specified before the command name - it will however still accept being specified after the command name (see examples below).

NOTE	There are some circumstances in which specifying the `--autoname` or `-a` option after the command name will result in the option not taking effect, which is why we recommend that you place it always before the

> command name.

When enabled, this feature allows commands that accept `String` names as their primary parameter (the parameter that you don't have to specify, e.g. the `instancename` parameter for the `create-instance` command) to generate a name. To automatically generate a name for your instances or resources, simply enable the `autoname` option and omit providing a name.

Specifically for the `create-instance` and `create-local-instance` commands, the `autoname` option can also be used to automatically resolve any name conflicts. When resolving a name conflict, Payara Server will simply append a new name to the existing name, like so: `instance1-BamboozledBarracuda`.

When creating an instance from the admin console, there is a new **Auto Name** checkbox present, which performs the same conflict resolution function as the `autoname` CLI option.

> **WARNING** This checkbox does not currently allow you to generate an instance name from scratch - it simply resolves any name conflicts.

It should be stressed that these options are optional and disabled by default - if you wish to retain the original behaviour of prompting for a name or refusing to create an instance due to a name conflict, no action is necessary.

Examples

Example 1 - Name generation:

```
asadmin -a create-instance --node localhost-domain1
-> BemusedCatfish created

asadmin create-local-instance --node localhost-domain1 --autoname
-> ConfusedCarp created

asadmin create-instance --node localhost-domain1
Please enter an instance name: instance1
-> instance1 created

asadmin create-instance --node localhost-domain1
Please enter an instance name: instance1
```

```
-> command failed: instance name already in use
```

Example 2 - Name conflict resolution:

```
asadmin --autoname create-instance --node localhost-domain1 instance1
-> instance1 created

asadmin create-instance --node localhost-domain1 --autoname instance1
-> instance1-BemusedCatfish created

asadmin create-instance --node localhost-domain1 instance1
-> command failed: instance name already in use
```

Logging

Capturing information on Payara Server runtime events.

Description

Logging is the process by which Payara Server captures data about events that occur during Payara Server operation. Payara Server components and application components generate logging data, which is saved in the server log, typically `domain-dir/logs/server.log`.

The server log is the first source of information if Payara Server problems occur.

The server log is rotated when the file reaches the specified size in bytes, or the specified time has elapsed. The file can also be rotated manually by using the `rotate-log` subcommand.

In addition to the server log, the `domain-dir/logs` directory contains two other kinds of logs:

- HTTP service access logs, located in the `/access` subdirectory
- Transaction service logs, located in the `/tx` subdirectory

Logging levels can be configured by using the Administration Console or the `set-log-levels` subcommand.

Additional properties can be set by using the Administration Console or by editing the

`logging.properties`file. The default `logging.properties` file is typically located in domain-dir`/config`.

See Also

- list-log-levels,
- rotate-log,
- set-log-levels
- asadmin-1m
- administering-the-logging-service in Payara Server General Administration section

Security

Secure and administer Payara Server applications.

Description

Security is about protecting data: how to prevent unauthorized access or damage to it in storage or transit. Payara Server has a dynamic, extensible security architecture based on the Jakarta EE standard. Built-in security features include cryptography, authentication and authorization, and public key infrastructure. Payara Server is built on the Java security model, which uses a sandbox where applications can run safely, without potential risk to systems or users.

See Also

- change-admin-password,
- change-master-password,
- create-auth-realm,
- create-file-user,
- create-message-security-provider,
- create-password-alias,
- create-ssl,

- delete-auth-realm,
- delete-file-user,
- delete-message-security-provider,
- delete-password-alias,
- delete-ssl,
- list-auth-realms,
- list-connector-security-maps,
- list-file-groups,
- list-file-users.
- asadmin-1m

Add-Instance-To-Deployment-Group

Adds a standalone instance to an existing deployment group.

Synopsis

```
asadmin [asadmin-options] add-instance-to-deployment-group [--help]
[--instance instanceName]
[--deploymentgroup deploymentGroupName]
```

Description

The `add-instance-to-deployment-group` subcommand will add the referenced instance to the referenced deployment group.

This command will only work with standalone instances (see types-of-payara-server-instances for more information).

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: `-?`: Displays the help text for the subcommand.

`--instance`: The name of the standalone instance that will be added to the deployment group. The command will fail if the name doesn't match to an existing standalone instance.

`--deploymentGroup`: The name of the deployment group to which the standalone instance will be added. The command will fail if the name doesn't match to an existing deployment group.

Examples

Example 1 Add an instance to a Deployment Group

This example adds an instance named `test-instance-1` to a deployment group named `test-dgroup`:

```
asadmin add-instance-to-deployment-group --instance=test-instance-1
--deploymentGroup=test-dgroup

Command add-instance-to-deployment-group executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-deployment-group,
- delete-deployment-group,
- list-deployment-groups
- remove-instance-from-deployment-group

Add-Library

Adds one or more library JAR files to Payara Server

Synopsis

ifeval: ["enterprise" == "community"]

```
asadmin [asadmin-options]
add-library
[--help]
[--type={common|ext|app|war}]
[--upload={false|true}]
[library-file-path ...]
```

endif: []

ifeval: ["enterprise" == "enterprise"]

```
asadmin [asadmin-options]
add-library
[--help]
[--type={common|ext|app}]
[--upload={false|true}]
[library-file-path ...]
```

endif: []

Description

The `add-library` subcommand adds one or more library archive files to Payara Server.

The `--type` option specifies the library type and the Payara Server directory to which the library is added.

The library-file-path operand is the path to the JAR file that contains the library to be added.To specify multiple libraries, specify multiple paths separated by spaces.

IMPORTANT	The library archive file is added to the DAS. For common and extension libraries, you must restart the DAS so the libraries are picked up by the server runtime.

To add the libraries to other server instances, synchronize the instances with the DAS by restarting them.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--type`: Specifies the library type and the Payara Server directory to which the library is added. Valid values are as follows:

+ `common`;; Adds the library files to the Common class loader directory, `domain-dir/lib`. This is the default. `ext`;; Adds the library files to the Java optional package directory, `domain-dir/lib/ext`. `app`;; Adds the library files to the application-specific class loader directory, `domain-dir/lib/applibs`.

ifeval: ["enterprise" == "community"] `war`;; Adds the library files to the war-packaged class loader directory, `domain-dir/lib/warlibs`. **endif**: []

+ For more information about these directories, see class-loaders in the Payara Server Application Development section.

`--upload`: Specifies whether the subcommand uploads the file to the DAS. In most situations, this option can be omitted.

+ NOTE: If a directory filepath is specified, this option is ignored.

+ Valid values are as follows:

+ `false`;; The subcommand does not upload the file and attempts to access the file through the specified file name. If the DAS cannot access the file, the subcommand fails.

+ For example, the DAS might be running as a different user than the administration user and does not have read access to the file. In this situation, the subcommand fails if the `--upload` option is `false`. `true`;; The subcommand uploads the file to the DAS

over the network connection.

+ The default value depends on whether the DAS is on the host where the subcommand is run or is on a remote host.

+ * If the DAS is on the host where the subcommand is run, the default is `false`. * If the DAS is on a remote host, the default is `true`.

Operands

library-file-path: The paths to the archive files that contain the libraries that are to be added. You can specify an absolute path or a relative path.
If the `--upload` option is set to `true`, this is the path to the file on the local client machine. If the `--upload` option is set to `false`, this is the path to the file on the server machine.

Examples

Example 1 Adding Libraries

This example adds the library in the archive file `mylib.jar` to the application-specific class loader directory on the default server instance.

```
asadmin add-library --type app /tmp/mylib.jar
Command add-library executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-libraries,
- remove-library,
- "Class Loaders"

Add-Resources

Creates the resources specified in an XML file.

Synopsis

```
asadmin [asadmin-options] add-resources [--help]
[--target target]
[--upload={false|true}] xml-file-name
```

Description

The `add-resources` subcommand creates the resources named in the specified XML file. The resources that can be created with this subcommand are listed in S ee Also in this help page.

The `--target` option specifies the target for which you are creating the resources. If this option specifies the domain, the resources are added only to the configuration of the domain administration server (DAS). If this option specifies any other target, the resources are added to the configuration of the DAS and references are added to the resources from the specified target.

The xml-file-name operand is the path to the XML file that contains the resources to be created. The `DOCTYPE` must be specified as `https://docs.azul.com/payara/appendix/schemas/payara-resources_1_8.dtd` in the `resources.xml` file.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which you are creating the resources. Valid values are as follows:
`server`;; Creates the resources for the default server instance `server` and is the default value. `domain`;; Creates the resources for the domain. cluster-name;; Creates the resources for every server instance in the cluster. instance-name;; Creates the resources for a particular Payara Server instance. **`--upload`**: Specifies whether the

subcommand uploads the file to the DAS. In most situations, this option can be omitted. Valid values are as follows:

`false`;; The subcommand does not upload the file and attempts to access the file through the specified file name. If the DAS cannot access the file, the subcommand fails.

For example, the DAS might be running as a different user than the administration user and does not have read access to the file. In this situation, the subcommand fails if the `--upload` option is `false`. `true`;; The subcommand uploads the file to the DAS over the network connection.

The default value depends on whether the DAS is on the host where the subcommand is run or is on a remote host.

* If the DAS is on the host where the subcommand is run, the default is `false`. * If the DAS is on a remote host, the default is `true`.

Operands

xml-file-name: The path to the XML file that contains the resources that are to be created. You can specify an absolute path, only the file name, or a relative path.

* If you specify an absolute path, the XML file can be anywhere. * If you specify only the file name, the XML file must reside in the domain-dir`/config` directory on the DAS host. This requirement must be met even if you run the subcommand from another host. * If you specify a relative path, the XML file must be in the relative directory.

An example XML file follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC
  "-//Payara.fish//DTD Payara Server 4 Resource Definitions//EN"
  "https://docs.azul.com/payara/appendix/schemas/payara-resources_1_8.dtd">
<resources>
 <jdbc-connection-pool name="SPECjPool" steady-pool-size="100"
   max-pool-size="150" max-wait-time-in-millis="60000"
   pool-resize-quantity="2" idle-timeout-in-seconds="300"
   is-isolation-level-guaranteed="true"
   is-connection-validation-required="false"
   connection-validation-method="auto-commit"
   fail-all-connections="false"
   datasource-classname="oracle.jdbc.pool.OracleDataSource">
  <property name="URL"
    value="jdbc:oracle:thin:@iasperfsol12:1521:specdb"/>
  <property name="User" value="spec"/>
  <property name="Password" value="spec"/>
```

```
   <property name="MaxStatements" value="200"/>
   <property name="ImplicitCachingEnabled" value="true"/>
  </jdbc-connection-pool>
  <jdbc-resource enabled="true" pool-name="SPECjPool"
    jndi-name="jdbc/SPECjDB"/>
</resources>
```

Examples

Example 1 Adding Resources

This example creates resources using the contents of the XML file `resource.xml`.

```
asadmin> add-resources resource.xml
Command : Connector connection pool jms/testQFactoryPool created.
Command : Administered object jms/testQ created.
Command : Connector resource jms/testQFactory created.
Command : Resource adapter config myResAdapterConfig created successfully
Command : JDBC connection pool DerbyPoolA created successfully.
Command : JDBC resource jdbc/__defaultA created successfully.
Command add-resources executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jdbc-connection-pool,create-jdbc-resource,
- create-jms-resource,
- create-jndi-resource,
- create-javamail-resource,
- create-custom-resource,
- create-connector-resource,
- create-connector-work-security-map,
- create-admin-object,
- create-resource-adapter-config

Add-To-Keystore

Adds the specified keystore bundle to a target domain keystore.

Synopsis

```
asadmin add-to-keystore [--help]
[--reload={false|true}]
[--domainName=name]
[--domainDir=directory]
[--node=nodeName]
[--nodeDir=directory]
[--listener=listenername]
[--target=instanceName]
[--file=<file-path>]
[alias]
```

Description

This command adds a certificate bundle (e.g. `.p12` or `.jks` file) to the target instance or listener's key store using the provided alias.

If the instance or listener is configured to use the default key store, the command will instead synchronize the instance with the DAS (under the assumption the certificate has been added to the default key store of the DAS), since any certificates added to the instance stores would be lost upon next synchronisation.

NOTE	This will overwrite an entry already present with the same alias. A certificate without a private key cannot be used by an HTTP listener and will generate a warning.

Options

`--help`: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are adding to its keystore. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--reload`**: Whether the HTTP listeners should be reloaded. Defaults to `false` **`--listener`**: The name of the HTTP or IIOP listener to add the certificate to. **`--domainname`**: The name of the domain where the target instance exists.

+ Defaults to `domain1` or the existing domain if only one exists. **`--domainDir`**: The

path to the directory containing the target domain.

+ Defaults to `as-install/glassfish/domains` **`--node`**: The name of the node where the target instance exists.

+ Defaults to `localhost-$domainname` **`--nodeDir`**: The path to the directory containing the target node.

+ Defaults to `as-install/glassfish/nodes`

`--file`: The bundle file to add to the target keystore.

Operands

alias: The alias name to store the certificate bundle in the keystore under.

Examples

Example 1 Add the specified P12 bundle under the `mycert` alias

```
asadmin add-to-keystore --file mycert.p12 mycert
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- "SSL Certificate Management",
- "`add-to-truststore`"

Add-To-Truststore

Adds the specified certificate to a target domain truststore.

Synopsis

```
asadmin add-to-truststore [--help]
[--reload={false|true}]
[--domainName=name]
[--domainDir=directory]
[--node=nodeName]
[--nodeDir=directory]
[--listener=listenername]
```

```
[--target=instanceName]
[--file=<file-path>]
[alias]
```

Description

This command adds a certificate (e.g. `.cer`/`.pem` file) to the target instance or listener's trust store or listener's trust store using the provided alias.

If the instance or listener is configured to use the default trust store, the command will instead synchronize the instance with the DAS (under the assumption the certificate has been added to the default trust store of the DAS), since any certificates added to the instance stores would be lost upon next synchronisation.

IMPORTANT	This will overwrite an entry already present with the same alias.

Options

`--help`: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are adding the certificate. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--reload`**: Whether the HTTP listeners should be reloaded. Defaults to `false` **`--listener`**: The name of the HTTP or IIOP listener to add the certificate to. **`--domainname`**: The name of the domain where the target instance exists.

+ Defaults to `domain1` or the existing domain if only one exists. **`--domainDir`**: The path to the directory containing the target domain.

+ Defaults to `as-install/glassfish/domains` **`--node`**: The name of the node where the target instance exists.

+ Defaults to `localhost-$domainname` **`--nodeDir`**: The path to the directory containing the target node.

+ Defaults to `as-install/glassfish/nodes`

`--file`: The certificate file to add to the target truststore.

Operands

alias: The alias name to store the certificate in the truststore under.

Examples

Example 1 Add the specified certificate under the `mycert` alias

```
asadmin add-to-truststore --file mycert.pem mycert
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- "SSL Certificate Management",
- "`add-to-keystore`"

Appclient

Launches the Application Client Container and invokes the client application typically packaged in the application JAR file.

Synopsis

```
appclient [client_application_classfile | -client client_application_jar]
[-mainclass main_class_name | -name display_name]
[-xml sun-acc.xml file] [-textauth]
[-targetserver host[:port][,host[:port]...]]
[-user username] [-passwordfile password_file]
[application-options]

appclient [jvm-options]
[-mainclass main_class_name | -name display_name]
[-xml client_config_xml_file] [-textauth]
[-targetserver host[:port][,host[:port]...]]
[-user username] [-passwordfile password_file]
class-selector [application-options]
```

Description

Use the `appclient` command to launch the Application Client Container and invoke a client application that is typically packaged in an application JAR file. The application client JAR file is specified and created during deployment by the Administration Console or the `asadmin deploy` command with the `--retrieve` option. You can

also retrieve the client JAR file using the `asadmin get-client-stubs` command.

The Application Client Container is a set of Java classes, libraries, and other files that are required to execute a first-tier application client program on a Virtual Machine for the Java platform (JVM machine). The Application Client Container communicates with the server using RMI-IIOP.

The client JAR file that is retrieved after deploying an application should be passed with the `-client` or `-jar` option when running the `appclient` utility. The client JAR file name is of the form app-name`Client.jar`. For multiple application clients in an EAR file, you must use the `-mainclass` or `-name` option to specify which client to invoke.

If the application client is a stand-alone module or the only client in an EAR file, the Application Client Container can find the client without using the `-mainclass` or `-name` options. If you provide a `-mainclass` or `-name` value that does not match what is in the client, the Application Client Container launches the client anyway but issues a warning that the selection did not match the information in the client. The warning also displays what the actual main class and name values are for the client.

Options

jvm-options: optional; you can set JVM options for the client application. These can be any valid `java` command options except `-client` or `-jar`. JVM options can be intermixed with other `appclient` command options as long as both types of options appear before the class-selector. **client_application_classfile**: optional; the file system pathname of the client application `.class` file. A relative pathname must be relative to the current directory. This class file must contain the `main()` method to be invoked by the Application Client Container.
If you use client_application_classfile and the class is dependent on other user classes, you must also set the classpath. You can either use the `-classpath` JVM option in the `appclient` command or set the `CLASSPATH` environment variable. For more information about setting a classpath, see Setting the Class Path, Oracle Solaris Version (`http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/classpath.html`) or Setting the Class Path, Windows Version

(`http://docs.oracle.com/javase/6/docs/technotes/tools/windows/classpath.html`). **`-client`**: optional; the name and location for the client JAR file. **`-mainclass`**: optional; the full classname of the main client application as specified in the `Main-Class` entry in the `MANIFEST.MF` file. Used for a multiple client applications. By default, uses the class specified in the `client jar`. For example, `com.example.test.AppClient`. **`-name`**: optional; the display name for the client application. Used for multiple client applications. By default, the display name is specified in the client jar `application-client.xml` file which is identified by the `display-name` attribute. **`-xml`**: optional if using the default domain, instance, and name (`sun-acc.xml`), otherwise it is required; identifies the name and location of the client configuration XML file. If not specified, defaults to the `sun-acc.xml` file in the domain-dir`/config` directory. **`-textauth`**: optional; used to specify using text format authentication when authentication is needed. **`-targetserver`**: optional; a comma-separated list of one or more server specifications for ORB endpoints. Each server specification must include at least the host. Optionally, a server specification can include the port as well. If the port is omitted from a server specification, the default value, `3700`, is used for that host. **`-user`**: optional; the application user who is authorized to have access to particular guarded components in the application, such as EJB components. **`-passwordfile`**: optional; specifies the name, including the full path, of a file that contains the password for application clients in the following format:

```
PASSWORD=appclient-password
```

If this option is omitted, the password is specified interactively at the command prompt.

CAUTION	Avoid specifying a password interactively at the command prompt. Such a password can be seen by users who can run commands to display running processes and the commands to start them, such as `ps`.

```
  For security reasons, a password that is specified as an environment variable is
not read by the `appclient` utility.
*class-selector*:
  required; you must specify the client application class using one of the
```

```
following class selectors. +
  `-jar` jar-file;;
    the name and location of the client JAR file. The application client JAR file
is specified and created during deployment by the `asadmin deploy` command. If
specified, the `-classpath` setting is ignored in deference to the `Class-Path`
setting in the client JAR file's manifest.
  class-name;;
    the fully qualified name of the application client's main class. The
Application Client Container invokes the `main` method of this class to start the
client. For example, `com.example.test.AppClient`. +
    If you use class-name as the class selector, you must also set the classpath.
You can either use the `-classpath` JVM option in the `appclient` command or set
the `CLASSPATH` environment variable. For   more information about setting a
classpath, see Setting the Class Path, Oracle Solaris Version
(`http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/classpath.html`) or
Setting the Class Path, Windows Version
(`http://docs.oracle.com/javase/6/docs/technotes/tools/windows/classpath.html`).
*application-options*:
  optional; you can set client application arguments.
```

Examples

Example 1 Using the `appclient` command

```
appclient -xml sun-acc.xml -jar myclientapp.jar scott sample
```

Where: `sun-acc.xml` is the name of the client configuration XML file, `myclientapp.jar` is the client application `.jar` file, and `scott` and `sample` are arguments to pass to the application. If `sun-acc.xml` and `myclientapp.jar` are not in the current directory, you must give the absolute path locations; otherwise the relative paths are used. The relative path is relative to the directory where the command is being executed.

Attributes

See `attributes` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Unstable

See Also

- asadmin-1m,
- get-client-stubs,

- package-appclient

Asadmin-Recorder-Enabled

Returns whether the Asadmin Recorder service is enabled .

Synopsis

```
asadmin [asadmin-options] asadmin-recorder-enabled [--help]
```

Description

The `asadmin-recorder-enabled` command checks if the Asadmin Recorder feature is active in Payara Server, simply returning a true or false status.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1

Returns whether the Asadmin Recorder Service is enabled or not.

```
asadmin> asadmin-recorder-enabled
Asadmin Recorder Service is enabled
Command asadmin-recorder-enabled executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-asadmin-recorder-configuration,
- get-asadmin-recorder-configuration,
- enable-asadmin-recorder,

- disable-asadmin-recorder

Asadmin

Utility for performing administrative tasks for Payara Server.

Synopsis

```
asadmin
[--host host]
[--port port]
[--user admin-user]
[--passwordfile filename]
[--terse={true|false}]
[--extraterse={true|false}]
[--secure={false|true}]
[--echo={true|false}]
[--interactive={true|false}]
[--detach={true|false}]
[--autoname={true|false}]
[--help]
[subcommand [options] [operands]]
```

Description

Use the `asadmin` utility to perform administrative tasks for Payara Server. You can use this utility instead of the Administration Console interface.

Subcommands of the asadmin Utility

The subcommand identifies the operation or task that you are performing. Subcommands are case-sensitive. Each subcommand is either a local subcommand or a remote subcommand.

- A local subcommand can be run without a running domain administration server (DAS). However, to run the subcommand and have access to the installation directory and the domain directory, the user must be logged in to the machine that hosts the domain.
- A remote subcommand is always run by connecting to a DAS and running the subcommand there. A running DAS is required.

Asadmin Utility Options and Subcommand Options

Options control the behavior of the `asadmin` utility and its subcommands. Options are

also case-sensitive.

The `asadmin` utility has the following types of options:

- `asadmin` utility options. These options control the behavior of the `asadmin` utility, not the subcommand. The `asadmin` utility options may precede or follow the subcommand, but `asadmin` utility options after the subcommand are deprecated. All `asadmin` utility options must either precede or follow the subcommand. If `asadmin` utility options are specified both before and after the subcommand, an error occurs. For a description of the `asadmin` utility options, see the "Options" section of this help information.
- Subcommand options. These options control the behavior of the subcommand, not the `asadmin` utility. Subcommand options must follow the subcommand. For a description of a subcommand's options, see the help information for the subcommand.

A subcommand option may have the same name as an `asadmin` utility option, but the effects of the two options are different.

The `asadmin` utility options and some subcommand options have a long form and a short form.

- The long form of an option has two dashes (`--`) followed by an option word.
- The short form of an option has a single dash (`-`) followed by a single character.

For example, the long form and the short form of the option for specifying terse output are as follows:

- Long form: `--terse`
- Short form: `-t`

Most options require argument values, except Boolean options, which toggle to enable or disable a feature.

Operands of Asadmin Subcommands

Operands specify the items on which the subcommand is to act. Operands must follow the argument values of subcommand options, and are set off by a space, a tab, or double dashes (`--`). The `asadmin` utility treats anything that follows the subcommand options and their values as an operand.

Escape Characters in Options for the Asadmin Utility

Escape characters are required in options of the `asadmin` utility for the following types of characters:

- Meta characters in the UNIX operating system. These characters have special meaning in a shell. Meta characters in the UNIX operating system include:

  ```
  `\/,.!$%^&*|{}[]"'~;`.
  ```

  To disable these characters, use the backslash (`\`) escape character or enclose the entire command-line argument in single quote (`\'`) characters.

  The following examples illustrate the effect of escape characters on the`*` character. In these examples, the current working directory is the `domains` directory.

  - The following command, without the escape character, echoes all files in the current directory:

    ```
    prompt% echo *
    domain1 domain2
    ```

  - The following command, in which the backslash (`\`) escape character precedes the `*` character, echoes the `*` character:

    ```
    prompt% echo \*
    *
    ```

  - The following command, in which the `*` character is enclosed in single quote (`\'`) characters, echoes the `\*` character:

```
prompt% echo '*'
*
```

The escape character is also a special character in the UNIX operating system and in the Java language. Therefore, in the UNIX operating system and in multimode, you must apply an additional escape character to every escape character in the command line. This requirement does not apply to the Windows operating system.

For example, the backslash (`\`) UNIX operating system meta character in the option argument `Test\Escape\Character` is specified on UNIX and Windows systems as follows:

- On UNIX systems, each backslash must be escaped with a second backslash:

```
Test\\Escape\\Character
```

- On Windows systems, no escape character is required:

```
Test\Escape\Character
```

NOTE

In contexts where meta characters in the UNIX operating system are unambiguous, these characters do not require escape characters. For example, in the set subcommand, the value that is to be set is specified as `name=value`.
Because name can never include an equals sign, no escape character is required to disable the equals sign. Therefore, everything after the equals sign is an uninterpreted string that the `set` subcommand uses unchanged.

- Spaces. The space is the separator in the command-line interface. To distinguish a space in a command-line argument from the separator in the command-line interface, the space must be escaped as follows:
  - For the UNIX operating system in single mode and multimode, and for all

operating systems in multimode, use the backslash (`\`) escape character or enclose the entire command-line argument in single quote (`\'`) characters or double quote (") characters.

- For the Windows operating system in single mode, enclose the entire command-line argument in double quote (") characters.

- Option delimiters. The `asadmin` utility uses the colon character (`:`) as a delimiter for some options. The backslash (`\`) escape character is required if the colon is part of the following items:
  - A property
  - An option of the Virtual Machine for the Java platform (Java Virtual Machine or JVM machine)

  For example, the operand of the subcommand create-jvm-options specifies JVM machine options in the following format:

  ```
  (jvm-option-name[=jvm-option-value])
  [:jvm-option-name[=jvm-option-value]]*
  ```

  Multiple JVM machine options in the operand of the `create-jvm-options` subcommand are separated by the colon (`:`) delimiter. If jvm-option-name or jvm-option-value contains a colon, the backslash (`\`) escape character is required before the colon.

  The backslash (`\`) escape character is also required before a single quote (`\'`) character or a double quote (`"`) character in an option that uses the colon as a delimiter.

  When used without single quote (`\'`) characters, the escape character disables the option delimiter in the command-line interface.

  For the UNIX operating system in single mode and multimode, and for all operating systems in multimode, the colon character and the backslash character in an option that uses the colon as a delimiter must be specified as follows:

- To pass a literal backslash to a subcommand, two backslashes are required. Therefore, the colon (`:`) must be escaped by two backslashes (`\\`).
- To prevent a subcommand from treating the backslash as a special character, the backslash must be escaped. As a result, two literal backslashes (`\\`) must be passed to the subcommand. To prevent the shell from interpreting these backslashes as special characters, each backslash must be escaped. Therefore, the backslash must be specified by a total of four backslashes (`\\\\`).

  For the Windows operating system in single mode, a backslash (`\`) is required to escape the colon (`:`) and the backslash (`\`) in an option that uses the colon as a delimiter.

Instead of using the backslash (`\`) escape character, you can use the double quote ("") character or single quote (') character. The effects of the different types of quote characters on the backslash (`\`) character are as follows:

- Between double quote (") characters, the backslash (`\`) character is a special character.
- Between single quote (') characters, the backslash (`\`) character is not a special character.

Requirements for Using the `--secure` Option

The requirements for using the `--secure` option are as follows:

- The domain that you are administering must be configured for security.
- The `security-enabled` attribute of the `http-listener` element in the DAS configuration must be set to `true`.

  To set this attribute, use the `set` subcommand.

Server Restart after Creation or Deletion

When you use the `asadmin` subcommands to create or delete a configuration item, you must restart the DAS for the change to take effect. To restart the DAS, use the restart-

domain subcommand.

Help Information for Subcommands and the Asadmin Utility

To obtain help information for an `asadmin` utility subcommand, specify the subcommand of interest as the operand of the `help` subcommand. For example, to obtain help information for the start-domain subcommand, type:

```
asadmin help start-domain
```

If you run the `help` subcommand without an operand, this help information for the `asadmin` utility is displayed.

To obtain a listing of available `asadmin` subcommands, use the list-commands subcommand.

Options

`--host`: **`-H`**: The machine name where the DAS is running. The default value is `localhost`. **`--port`**: **`-p`**: The HTTP port or HTTPS port for administration. This port is the port in the URL that you specify in your web browser to manage the domain. For example, in the URL `http://localhost:4949`, the port is 4949.

+ The default port number for administration is 4848. **`--user`**: **`-u`**: The username of the authorized administrative user of the DAS.

+ If you have authenticated to a domain by using the `asadmin login` command, you need not specify the `--user` option for subsequent operations on the domain. **`--passwordfile`**: **`-W`**: Specifies the name, including the full path, of a file that contains password entries in a specific format.

+ CAUTION: Any password file created to pass as an argument by using the `--passwordfile` option should be protected with file system permissions. Additionally, any password file being used for a transient purpose, such as setting up SSH among nodes, should be deleted after it has served its purpose.

+ The entry for a password must have the `AS_ADMIN_` prefix followed by the password

name in uppercase letters, an equals sign, and the password.

+ The entries in the file that are read by the `asadmin` utility are as follows:

- `AS_ADMIN_PASSWORD`=administration-password
- `AS_ADMIN_MASTERPASSWORD`=master-password
  The entries in this file that are read by subcommands are as follows:
- `AS_ADMIN_NEWPASSWORD=\`new-administration-password\`` (read by the start-domain subcommand)
- `AS_ADMIN_USERPASSWORD=\`user-password\`` (read by the create-file-user subcommand)
- `AS_ADMIN_ALIASPASSWORD=\`alias-password\`` (read by the create-password-alias subcommand)
- `AS_ADMIN_MAPPEDPASSWORD=\`mapped-password\`` (read by the create-connector-security-map subcommand)
- `AS_ADMIN_SSHPASSWORD=\`sshd-password\`` (read by the create-node-ssh, install-node, install-node-ssh, and update-node-ssh subcommands)
- `AS_ADMIN_SSHKEYPASSPHRASE=\`sshd-passphrase\`` (read by the create-node-ssh, install-node, install-node-ssh, and update-node-ssh subcommands)
- `AS_ADMIN_JMSDBPASSWORD=` jdbc-user-password (read by the configure-jms-cluster subcommand)

  These password entries are stored in clear text in the password file. To provide additional security, the `create-password-alias` subcommand can be used to create aliases for passwords that are used by remote subcommands. The password for which the alias is created is stored in an encrypted form. If an alias exists for a password, the alias is specified in the entry for the password as follows:

  ```
  AS_ADMIN_password-name=${ALIAS=password-alias-name}
  ```

  For example:

```
AS_ADMIN_SSHPASSWORD=${ALIAS=ssh-password-alias}
AS_ADMIN_SSHKEYPASSPHRASE=${ALIAS=ssh-key-passphrase-alias}
```

In domains that do not allow unauthenticated login, all remote subcommands must specify the administration password to authenticate to the DAS. The password can be specified by one of the following means:

- Through the `--passwordfile` option
- Through the link:Technical Documentation/Payara Server Documentation/Command Reference/login#login[`login`] subcommand
- Interactively at the command prompt

The `login` subcommand can be used to specify only the administration password. For other passwords that remote subcommands require, use the `--passwordfile` option or specify them at the command prompt.

After authenticating to a domain by using the `asadmin login` command, you need not specify the administration password through the `--passwordfile` option for subsequent operations on the domain. However, only the `AS_ADMIN_PASSWORD` option is not required. You still must provide the other passwords, for example, `AS_ADMIN_USERPASSWORD`, when required by individual subcommands, such as link:Technical Documentation/Payara Server Documentation/Command Reference/update-file-user#update-file-user[`update-file-user`].

For security reasons, a password that is specified as an environment variable is not read by the `asadmin` utility.

The master password is not propagated on the command line or an environment variable, but can be specified in the file that the `--passwordfile` option specifies.

The default value for `AS_ADMIN_MASTERPASSWORD` is `changeit`. **`--terse`**: **`-t`**: If true, output data is very concise and in a format that is optimized for use in scripts instead of for reading by humans. Typically, descriptive text and detailed status messages are also omitted from the output data.

Default is `false`. **`--extraterse`**: **`-T`**:

NOTE	Currently, this option only works with the `create-instance` and `create-local-instance` commands.

If `true`, the `create-instance` and `create-local-instance` commands should only return the name of the instance created.

See the-extra-terse-option for more information.

IMPORTANT	The `extraterse` option implicitly enables the `terse` option - this is true for **all** commands (not just `create-instance` and `create-local-instance`).

Defaults to `false`. **`--secure`**: **`-s`**: If set to `true`, uses SSL/TLS to communicate with the DAS.

The default is `false`. **`--echo`**: **`-e`**: If set to `true`, the command-line statement is echoed on the standard output. The default is `false`. **`--interactive`**: **`-I`**: If set to `true`, only the required options are prompted.

The default depends on how the `asadmin` utility is run:

- If the `asadmin` utility is run from a console window, the default is `true`.
- If the `asadmin` utility is run without a console window, for example, from within a script, the default is `false`. **`--detach`**: If set to `true`, the specified `asadmin` subcommand is detached and executed in the background in detach mode. The default value is `false`.

The `--detach` option is useful for long-running subcommands and enables you to execute several independent subcommands from one console or script.

The `--detach` option is specified before the subcommand. For example, in single mode, `asadmin --detach` subcommand.

Job IDs are assigned to subcommands that are started using `asadmin --detach`. You can use the list-jobs subcommand to view the jobs and their job IDs, the attach subcommand to reattach to the job and view its status and output, and the configure-managed-jobs subcommand to configure how long information about the jobs is kept.

`--autoname`: **`-a`**: If `true`, this option allows the specified `asadmin` subcommand that accept `String` names as their primary parameter (the parameter that you don't have to specify, e.g. the `instancename` parameter for the `create-instance` command) to generate a name.

+ Specifically for the `create-instance` and `create-local-instance` commands, the `autoname` option can also be used to automatically resolve any name conflicts. See auto-naming for more information.

+ Default to `false`. **`--help`**: **`-?`**: Displays the help text for the `asadmin` utility.

Examples

Example 1 Running an `asadmin` Utility Subcommand in Single Mode

This example runs the list-applications subcommand in single mode. In this example, the default values for all options are used.

The example shows that the application `hello` is deployed on the local host.

```
asadmin list-applications
hello <web>

Command list-applications executed successfully.
```

Example 2 Specifying an `asadmin` Utility Option With a Subcommand

This example specifies the `--host asadmin` utility option with the list-applications` subcommand in single mode. In this example, the DAS is running on the host `srvr1.example.com`.

The example shows that the applications `basic-ezcomp`, `scrumtoys`, `ejb31-war`,

and `automatic-timer-ejb` are deployed on the host `srvr1.example.com`.

```
asadmin --host srvr1.example.com list-applications

basic-ezcomp <web>
scrumtoys <web>
ejb31-war <ejb, web>
automatic-timer-ejb <ejb>

Command list-applications executed successfully.
```

Example 3 Specifying an `asadmin` Utility Option and a Subcommand Option

This example specifies the `--host asadmin` utility option and the `--type` subcommand option with the `list-applications` subcommand in single mode. In this example, the DAS is running on the host `srvr1.example.com` and applications of type `web` are to be listed.

```
asadmin --host srvr1.example.com list-applications --type web

basic-ezcomp <web>
scrumtoys <web>
ejb31-war <ejb, web>

Command list-applications executed successfully.
```

Example 4 Escaping a Command-Line Argument With Single Quote Characters

The commands in this example specify the backslash (`\`) UNIX operating system meta character and the colon (`:`) option delimiter in the property value `c:\extras\pmdapp`.

For the UNIX operating system in single mode and multimode, and for all operating systems in multimode, the backslash (`\`) is required to escape the backslash (`\`) meta character and the colon (:) option delimiter:

```
asadmin deploy --property extras.home='c\:\\extras\\pmdapp' pmdapp.war

Application deployed with name pmdapp.
Command deploy executed successfully
```

For the Windows operating system in single mode, the single quote (`'`) characters eliminate the need for other escape characters:

```
asadmin deploy --property extras.home='c:\extras\pmdapp' pmdapp.war

Application deployed with name pmdapp.
Command deploy executed successfully
```

Example 5Specifying a UNIX Operating System Meta Character in an Option

The commands in this example specify the backslash (`\`) UNIX operating system meta character in the option argument `Test\Escape\Character`.

For the UNIX operating system in single mode and multimode, and for all operating systems in multimode, the backslash (`\`) is required to escape the backslash (`\`) meta character:

```
asadmin --user admin --passwordfile gfpass create-jdbc-connection-pool
--datasourceclassname sampleClassName --description Test\\Escape\\Character
sampleJDBCConnectionPool
```

For the Windows operating system in single mode, no escape character is required:

```
asadmin --user admin --passwordfile gfpass create-jdbc-connection-pool
--datasourceclassname sampleClassName --description Test\Escape\Character
sampleJDBCConnectionPool
```

Example 6 Specifying a Command-Line Argument That Contains a Space

The commands in this example specify spaces in the operand `C:\Documents and Settings\gfuser\apps\hello.war`.

For all operating systems in single mode or multimode, the entire operand can be enclosed in double quote (`"`) characters:

```
asadmin deploy "C:\Documents and Settings\gfuser\apps\hello.war"

Application deployed with name hello.
Command deploy executed successfully.
```

For the UNIX operating system in single mode and multimode, and for all operating systems in multimode, the entire command-line argument can be enclosed in single

quote (`'`) characters:

```
asadmin deploy 'C:\Documents and Settings\gfuser\apps\hello.war'

Application deployed with name hello.
Command deploy executed successfully.
```

Alternatively, for the UNIX operating system in single mode and multimode, and for all operating systems in multimode, the backslash (`\`) escape character can be used before each space in the operand. In this situation, the backslash (`\`) escape character is required before each backslash in the operand:

```
asadmin deploy C:\\Documents\ and\ Settings\\gfuser\\apps\\hello.war

Application deployed with name hello.
Command deploy executed successfully.
```

Example 7 Specifying a Meta Character and an Option Delimiter Character in a Property

The commands in this example specify the backslash (<code>\</code>) UNIX operating system meta character and the colon (:) option delimiter character in the <code>--property`option of the create-jdbc-connection-pool subcommand.

The name and value pairs for the `--property` option are as follows:

```
user=dbuser
passwordfile=dbpasswordfile
DatabaseName=jdbc:h2
server=http://localhost:9092
```

For the UNIX operating system in single mode and multimode, and for all operating systems in multimode, a backslash (`\`) is required to escape the colon (`:`) and the backslash (`\`):

```
asadmin --user admin --passwordfile gfpass create-jdbc-connection-pool
--datasourceclassname org.h2.jdbcx.JdbcDataSource --property user
=dbuser:passwordfile=dbpasswordfile:DatabaseName=jdbc\\:h2:server=http\\://localhos
t\\:9092 h2-pool
```

Alternatively, the entire argument to the `--property` option can be enclosed in single

quote (') characters:

```
asadmin --user admin --passwordfile gfpass create-jdbc-connection-pool
--datasourceclassname org.h2.jdbcx.JdbcDataSource --property
'user=dbuser:passwordfile=dbpasswordfile:DatabaseName="jdbc:h2":server="http://loca
lhost:9092"' h2-pool
```

For the Windows operating system in single mode, a backslash (`\`) is required to escape only the colon (`:`), but not the backslash (`\`):

```
asadmin --user admin --passwordfile gfpass create-jdbc-connection-pool
--datasourceclassname org.h2.jdbcx.JdbcDataSource --property user-
dbuser:passwordfile-dbpasswordfile:DatabaseName=jdbc\:h2:server=http\://localhost
\:9092 h2-pool
```

For all operating systems, the need to escape the colon (`:`) in a value can be avoided by enclosing the value in double quote characters or single quote characters:

```
asadmin --user admin --passwordfile gfpass create-jdbc-connection-pool
--datasourceclassname org.h2.jdbcx.JdbcDataSource --property user
=dbuser:passwordfile=dbpasswordfile:DatabaseName=\"jdbc:h2\":server=\"http://localh
ost:9092\" h2-pool
```

Example 8 Specifying an Option Delimiter and an Escape Character in a JVM Machine Option

The commands in this example specify the following characters in the `-Dlocation=c:\payara\appserver` JVM machine option:

- The colon (`:`) option delimiter
- The backslash (`\`) escape character

For the UNIX operating system in single mode and multimode, and for all operating systems in multimode, these characters must be specified as follows:

- To pass a literal backslash to a subcommand, two backslashes are required. Therefore, the colon (`:`) must be escaped by two backslashes (`\\`).
- To prevent the subcommand from treating the backslash as a special character, the

backslash must be escaped. As a result, two literal backslashes (`\\`) must be passed to the subcommand. To prevent the shell from interpreting these backslashes as special characters, each backslash must be escaped. Therefore, the backslash must be specified by a total of four backslashes (`\\\\`).

The resulting command is as follows:

```
asadmin create-jvm-options --target test-server -e -Dlocation=c\\:\\\\payara
\\\\appserver
```

For the Windows operating system in single mode, a backslash (`\`) is required to escape the colon (`:`) and the backslash (`\`):

```
asadmin create-jvm-options --target test-server -e -Dlocation=c\:\\payara
\\appserver
```

Example 9 Specifying an Option That Contains an Escape Character

The commands in this example specify the backslash (`\`) character and the double quote (`"`) characters in the `description` argument.

For the UNIX operating system in single mode and multimode, and for all operating systems in multimode, a backslash (`\`) is required to escape the double quote character (`"`) and the backslash (`\`):

```
asadmin set-web-env-entry --name="Hello User" --type=java.lang.String --value
=techscribe --description=\"Hello\\App\"\\authentication hello
```

For the Windows operating system in single mode, a backslash (`\`) is required to escape only the double quote (`"`), but not the backslash (`\`):

```
asadmin set-web-env-entry --name="Hello User" --type=java.lang.String --value
=techscribe --description=\"Hello\App\"\authentication hello
```

Environment Variables

Environment variables modify the default values of `asadmin` utility options as shown in the following table.

Environment Variable	`asadmin` Utility Option
`AS_ADMIN_TERSE`	`--terse`
`AS_ADMIN_EXTRATERSE`	`--extraterse`
`AS_ADMIN_ECHO`	`--echo`
`AS_ADMIN_INTERACTIVE`	`--interactive`
`AS_ADMIN_HOST`	`--host`
`AS_ADMIN_PORT`	`--port`
`AS_ADMIN_SECURE`	`--secure`
`AS_ADMIN_USER`	`--user`
`AS_ADMIN_PASSWORDFILE`	`--passwordfile`
`AS_ADMIN_HELP`	`--help`

See Also

- attach,
- configure-jms-cluster,
- configure-managed-jobs,
- create-connector-security-map,
- create-file-user,
- create-jdbc-connection-pool,
- create-jvm-options,
- create-node-ssh,
- create-password-alias,
- deploy,
- install-node,
- install-node-ssh,
- list-applications,
- list-commands,
- list-jobs,

- login,
- restart-domain,
- set,
- set-web-env-entry,
- start-domain,
- update-file-user,
- update-node-ssh

Attach

Attaches to subcommands that were started using `asadmin --detach` or that contain progress information.

Synopsis

```
asadmin [asadmin-options] attach [--help]
job_id
```

Description

The `attach` subcommand attaches to subcommands that were started using the `asadmin` utility option `--detach` or that contain progress information. The `--detach` option detaches long-running subcommands and executes them in the background in detach mode.

Job IDs are assigned to the subcommands (jobs), and can be used to view the status of a job and its output. Use the list-jobs subcommand to view the jobs and their job IDs, and the configure-managed-jobs subcommand to configure how long information about the jobs is kept.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the link:Technical Documentation/Payara Server Documentation/Command

Reference/asadmin#asadmin-1m[`asadmin`] help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

job_id: The ID of the job for which you want to view status and output.

Examples

Example 1: Attaching to a Subcommand and Checking Its Status

This example attaches to the `deploy` subcommand with a job ID of `20` and shows that the job is finished. If a subcommand is still in progress, the output displays the current status, for example, `64%: Uploading bits`.

```
asadmin> attach 20
Finished execution of deploy
Command attach executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- configure-managed-jobs,
- list-jobs

Backup-Domain

Performs a backup on the domain.

Synopsis

```
asadmin [asadmin-options] backup-domain [--help]
[--long[={false|true}]]
[--description description-text]
[--domaindir domain-root-dir]
[--backupdir backup-directory]
[--backupconfig backup-config-name]
[domain_name]
```

Description

The `backup-domain` subcommand backs up files under the named domain.

To complete the command, however, **the domain must be stopped first**. This will mean that any application deployed to the DAS is unavailable while the backup takes place.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--long`: **`-l`**: Displays detailed information about the backup operation.

+ The default value is `false`.

`--description`: Specifies a description to store in the backup file. The description is displayed as part of the information about a backup file.

+ The default value has this form:

+

```
domain-name_YYYY_MM_DD_backup-number
```

`--domaindir`: Specifies the domain root directory, the parent directory of the domain to back up.

+ The default value is `as-install/domains`.

`--backupdir`: Specifies the directory under which the backup file is to be stored.

+ The default value is `as-install/domains/domain-dir/backups`. If the domain is not in the default location, the location is `domain-dir/backups`.

Operands

domain-name: Specifies the name of the domain to be backed up.

+ This operand is optional if only one domain exists in the Payara Server installation.

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-backups,
- restore-domain

Capture-Schema

Stores the database metadata (schema) in a file for use in mapping and execution.

Synopsis

```
capture-schema -username name -password password
[-dburl url] [-driver jdbc_driver_classname]
[-schemaname schemaname] [-table tablename]
-out filename]
```

Description

Stores the database metadata (schema) in a file.

Run `capture-schema` as the same database user that owns the table(s), and use that same username with the `-username` option (and `-schemaname`, if required).

When running `capture-schema` against an Oracle database, you should grant the database user running the `capture-schema` command the `ANALYZE ANY TABLE` privilege.

Options

`-username`: user name for authenticating access to a database. **`-password`**: password for accessing the selected database. **`-dburl`**: JDBC URL required by the driver for accessing a database. **`-driver`**: JDBC driver classname in your `CLASSPATH`. **`-schemaname`**: name of the user schema being captured. If not specified, the default will capture metadata for all tables from all the schemas accessible to this user.

Specifying this parameter is highly recommended.
The specified schema name must be uppercase. `--table`: name of a table; multiple table names can be specified. If no table is specified, all the tables in the database or named schema are captured.
The specified table name or names are case sensitive. Be sure to match the case of the previously created table names. `--out`: name of the output file. This option is required. If the specified output file does not contain the .dbschema suffix, it will be appended to the filename.

Examples

Example 1 Using the `capture-schema` command

```
capture-schema --username cantiflas --password enigma
--dburl jdbc:oracle:thin:@sadbuttrue:1521:ora817
--driver oracle.jdbc.driver.OracleDriver
--schemaname CANTIFLAS --out cantiflas.dbschema
```

Where: `sun-acc.xml` is the name of the client configuration XML file, `myclientapp.jar` is the client application `.jar` file, and `scott` and `sample` are arguments to pass to the application. If `sun-acc.xml` and `myclientapp.jar` are not in the current directory, you must give the absolute path locations; otherwise the relative paths are used. The relative path is relative to the directory where the command is being executed.

See Also

- asadmin-1m

Change-Admin-Password

Changes the administrator password.

Synopsis

```
asadmin [asadmin-options] change-admin-password [--help]
[--domaindir domain-root-dir [--domain_name domain-name]]
```

Description

The `change-admin-password` subcommand modifies the administrator password. The `change-admin-password` subcommand is interactive because the subcommand prompts the user for the old administrator password, for the new administrator password, and for confirmation of the new administrator password. The new password must contain at least 8 characters.

If the only user is an anonymous user without a password, this subcommand fails.

If a blank password is provided, this subcommand fails if secure administration is enabled.

For security purposes, create a password-protected user account with administrator privileges. To create this account, use the create-file-user or the Administration Console. After creating this user account, remove the anonymous user to restrict access to Payara Server settings.

If more than one administrator is configured for Payara Server, you must run the `asadmin` command with the `--user` option to change the password for that user. For more information, see the examples in this help page.

This subcommand is supported in local mode and remote mode. If the `--domaindir` or `--domain_name` option is specified, the `change-admin-password` subcommand operates in local mode. If neither option is specified, the `change-admin-password` subcommand first attempts to operate in remote mode. If neither option is specified and the DAS is not running, the `change-admin-password` subcommand operates in local mode, using the default values for the `--domaindir` and `--domain_name` options.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--domaindir`**: Specifies the parent directory of the domain specified in the `--domain_name` option. When this option is used, the `change-admin-password`

subcommand operates in local mode. `--domain_name`: Specifies the domain of the admin user.

This option is not required if the directory specified by the `--domaindir` option contains only one domain.

Examples

Example 1 Changing the Administrator Password For a Single User in Multimode

```
asadmin --user admin
asadmin> change-admin-password
Please enter the old admin password>
Please enter the new admin password>
Please enter the new admin password again>
Command change-admin-password executed successfully.
```

Example 2 Changing the Administrator Password For a Single User in Single Mode

```
asadmin --user admin change-admin-password
Please enter the old admin password>
Please enter the new admin password>
Please enter the new admin password again>
Command change-admin-password executed successfully.
```

Exit Status

0: command executed successfully **1**: command failed

See Also

- asadmin-1m
- create-file-user,
- delete-password-alias,
- list-password-aliases,
- update-password-alias

Change-Master-Broker

Changes the master broker in a Message Queue cluster providing JMS services for a Payara Server cluster.

Synopsis

```
asadmin [asadmin-options] change-master-broker [--help]
clustered-instance-name
```

Description

The `change-master-broker` subcommand changes the master broker in a Message Queue cluster that is the JMS provider for a Payara Server cluster. By default, the master broker is the one associated with the first instance configured in the Payara Server cluster.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

clustered-instance-name: The name of the server instance whose Message Queue broker is to become the master broker of the Message Queue cluster. This server instance must be an instance in a Payara Server cluster.

Examples

Example 1 Changing the master broker

The following subcommand changes the Message Queue master broker to the one for the `clustinst3` clustered instance.

```
asadmin> change-master-broker clustinst3
Command change-master-broker executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m

Change-Master-Password

Changes the master password.

Synopsis

```
asadmin [asadmin-options] change-master-password [--help]
[--nodedir node-dir] [--domaindir domain-dir]
[--savemasterpassword={false|true}] [--masterpasswordlocation filepath]
[domain-name|node-name]
```

Description

The `change-master-password` subcommand is used to modify the master password. The `change-master-password` subcommand is interactive in that the user is prompted for the old master password, as well as the new master password. This subcommand will not work unless the server is stopped. In a distributed environment, this command must run on each machine in the domain.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--nodedir`**: The name of the directory containing the node instance for which the password will be changed. If this option is omitted, the change is applied to the entire domain. **`--domaindir`**: The name of the domain directory used for this operation. By default, the `--domaindir` option is `$AS_DEF_DOMAINS_PATH`, which is an environment variable defined in the file `asenv.bat` or `asenv.conf`.
`--savemasterpassword`: This option indicates whether the master password should be written to the file system. This is necessary so that the start-domain command can start the server without having to prompt the user.
The default is `false`. **`--masterpasswordlocation`**: This is the absolute filepath that the master password will be written to. This parameter is only active when `--savemasterpassword` is active.
If this parameter is unused, then the master password will be saved to the default location.

CAUTION	Saving the master password on disk is extremely insecure and should be avoided.

NOTE	If the `--savemasterpassword` option is not set, the master password file, if it exists, will be deleted.

Operands

domain-name|node-name: This name of the domain or node for which the password will be changed. If there is only a single domain, this is optional.

Examples

Example 1 Changing the Master Password

This example shows how to changed the master password for the `domain44ps` domain.

```
asadmin>change-master-password domain44ps
Please enter the new master password>
Please enter the new master password again>
Master password changed for domain44ps
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- delete-password-alias,
- list-password-aliases,
- start-domain,
- update-password-alias

Clean-Jbatch-Repository

Deletes records of Jakarta Batch job executions.

Synopsis

```
asadmin [asadmin-options] clean-jbatch-repository
[--status={COMPLETED|ALL}]
[--days] days
[job_name]
```

Description

The `clean-jbatch-repository` subcommand will remove the history of all job records related to an specific Jakarta Batch execution job.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--status`**: The status of batch jobs executions to which delete the records of. Valid values are as follows:
`COMPLETED`;; Only job executions that successfully completed will be deleted. `ALL`;; All job executions will be deleted, regardless of status. **`--days`**: The number of days previous that the end time of the job execution must be to get deleted.

+ Defaults to `1` day

Operands

job_name: The name of the Jakarta Batch job that will be cleared in the repository.

Examples

The following command will delete all job records for the `testjob` job that were executed before 7 days ago.

```
asadmin > clean-jbatch-repository --status=ALL --days=7 testjob
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m

- list-batch-jobs,
- list-batch-job-executions,
- list-batch-job-steps,
- list-batch-runtime-configuration,
- purge-jbatch-repository

Clear-Cache

Clears a Hazelcast or JCache IMap.

Synopsis

```
asadmin [asadmin-options] clear-cache
[--help]
[--target target]
[--cachename cachename]
```

Description

This command can be used to clear a Hazelcast or JCache IMap.

WARNING	Use of this command removes all data from a cache and therefore can result in unexpected behaviour if a system cache is cleared. Clearing a cache containing web session data will result in all users losing their sessions, for example.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are executing this operation. Valid values are:
`server`;; Applies to the default server instance. This is the default value.
instance_name;; Applies to a specified server instance. `--cachename`: Name of the cache/map to clear.

Examples

Example 1 Clears a Hazelcast or JCache IMap

```
asadmin> clear-cache --cachename payara.instance.store
Cleared Cache payara.instance.store
Command clear-cache executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Domain Data Grid",
- "`list-hazelcast-cluster-members`",
- "`list-hazelcast-members`",
- "`restart-hazelcast`",
- "`list-cache-keys`",
- "`list-caches`",
- "`set-hazelcast-configuration`",
- "`get-hazelcast-configuration`",
- "`generate-encryption-key`"

Collect-Diagnostics

Collects data used for the diagnostics and troubleshooting of Payara Server incidents.

Synopsis

```
asadmin [asadmin-options] collect-diagnostics [--help]
[--domainDir=directory]
[--nodeDir=directory]
[--target=target]
[--dir=<file-path>]
[--serverLog={false|true}]
[--accessLog={false|true}]
[--notificationLog={false|true}]
[--domainXml={false|true}]
[--threadDump={false|true}]
[--heapDump={false|true}]
[--jvmReport={false|true}]
[--obfuscate={true|false}]
```

```
[domainName]
```

Description

This command will collect useful information for diagnosing incidents within a Payara Server domain. It can collect from both remote and local instances.

Among the data collected is the following information:

- `domain.xml` configuration file
- Server logs
- Access logs
- Notification logs
- A JVM thread dump.
- A JVM Heap Dump
- JVM Report

NOTE	The data collected is stored in a ZIP file in your user home directory, the output directory by default.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which data is collected. Valid values are:
deployment_group_name;; Applies to every server instance in the specified deployment group. cluster_name;; Applies to every server instance in the specified cluster. instance_name;; Applies to a specific sever instance. **`--domainDir`**: The path to the directory containing the target domain.

+ Defaults to `as-install/glassfish/domains` **`--nodeDir`**: The path to the directory containing the target node.

+ Defaults to `as-install/glassfish/nodes` **`--dir`**: The directory path where the ZIP file will be stored. Defaults to `${user.home}` **`--domainXml`**: Whether to collect the `domain.xml` configuration file. Defaults to `true`. **`--serverLog`**: Whether to collect server logs. Defaults to `true` **`--accessLog`**: Whether to collect access logs. Defaults to `true` **`--notificationLog`**: Whether to collect notification logs. Defaults to `true` **`--heapDump`**: Whether to collect a heap dump. Defaults to `true` **`--threadDump`**: Whether to collect a thread dump. Defaults to `true` **`--jvmReport`**: Whether to collect the JVM report. Defaults to `true` **`--obfuscate`**: Whether to obfuscate data in `domain.xml`. Defaults to `true`

Operands

domainName: The name of the domain that will be scanned.

+ Defaults to `domain1`.

Examples

Example 1 Collect Diagnostics

The following example shows how to collect the data of all instances that belong to the `deploymentGroup1` deployment group in the `mydomain` domain:

```
asadmin collect-diagnostics --target deploymentGroup1 mydomain

Directory selected /home/user/payara/diagnostics/mydomain-2024-04-15T11-42-36Z
Collecting domain.xml from instance2
Attempting to create missing path at /home/user/payara/diagnostics/mydomain-2024-
04-15T11-42-36Z/deploymentGroup1/instance1
Collecting logs from instance2
Collecting jvm report from instance2
Collecting thread dump from instance2
Collecting Heap Dump from instance2
This version of Payara does not support heap dump generation.
Collecting domain.xml from instance1
Attempting to create missing path at /home/user/payara/diagnostics/mydomain-2024-
04-15T11-42-36Z/deploymentGroup1/instance1
Collecting logs from instance1
Collecting jvm report from instance1
Collecting thread dump from instance1
Collecting Heap Dump from instance1

Command collect-diagnostics executed successfully.
```

Example 2 Collect only a Heap Dump

The following example shows how to just collect a Heap dump of all instances in the `mydomain` domain:

```
asadmin collect-diagnostics --target deploymentGroup1 --threadDump=false
--domainXml=false --jvmReport=false --serverLog=false mydomain

Directory selected /home/user/payara/diagnostics/mydomain-2024-04-15T11-42-36Z
Collecting domain.xml from instance2
Attempting to create missing path at /home/user/payara/diagnostics/mydomain-2024-
04-15T11-42-36Z/deploymentGroup1/instance1
Collecting Heap Dump from instance2
This version of Payara does not support heap dump generation.
Collecting domain.xml from instance1
Attempting to create missing path at /home/user/payara/diagnostics/mydomain-2024-
04-15T11-42-36Z/deploymentGroup1/instance1
Collecting Heap Dump from instance1

Command collect-diagnostics executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- "Diagnostics Tool"

Collect-Log-Files

Creates a ZIP archive of all available log files.

Synopsis

```
asadmin [asadmin-options] collect-log-files [--help]
[--target target]
[--retrieve={false|true}] [retrievefilepath]]
```

Description

The `collect-log-files` subcommand collects all available log files for the domain administration server (DAS), the specified cluster, or the specified Payara Server instance and creates a single ZIP archive of the log files. If a cluster is specified, the ZIP archive also contains the log file for the DAS.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--retrieve`**: Specifies whether the ZIP archive is created in a directory other than the default directory.
By default the ZIP archive is created in the `domain-dir/collected-logs` directory. The ZIP file names are constructed from the specified target and timestamp, as follows:

+

```
log_yyyy-mm-dd_hh-min-sec.zip
```

```
  Possible values are as follows: +
`false`;;
    The ZIP archive will be created in the default directory. If omitted, the `--
retrieve` option defaults to `false`.
`true`;;
    The ZIP archive will be created in the directory that the retrievefilepath
operand specifies. If retrievefilepath is omitted, the ZIP archive will be created
in the default directory.
*`--target`*:
  Specifies the target for which log files will be collected. +
  Possible values are as follows: +
`server`;;
    The log files will be collected for the DAS (default).
`instance-name`;;
    The log files will be collected for the specified Payara Server instance.
`cluster-name`;;
    The log files will be collected for the specified cluster and the DAS.
```

Operands

retrievefilepath: The name of the directory in which the ZIP archive will be created. If this operand is omitted, the ZIP archive will be created in the default directory. If the `--retrieve` option is `false`, this operand is ignored.

Examples

Example 1 Collecting Log Files for the Default Server

This example generates a ZIP archive from the log files for the default server.

```
asadmin> collect-log-files
Created Zip file under /space/gfv3/v3setup/payara7/glassfish/domains/domain1/\
collected-logs/log_2010-12-15_15-46-23.zip.
```

```
Command collect-log-files executed successfully.
```

Example 2 Collecting Log Files for a Cluster

This example generates a ZIP archive from the log files for a cluster named `cluster1` and the two server instances running in the cluster.

```
asadmin> collect-log-files --target cluster1
Log files are downloaded for instance1.
Log files are downloaded for instance2.
Created Zip file under /space/gfv3/v3setup/payara7/glassfish/domains/domain1/\
collected-logs/log_2010-12-15_15-54-06.zip.
Command collect-log-files executed successfully.
```

Example 3 Collecting Log Files in a Directory Other Than the Default for a Cluster

This example generates a ZIP archive from the log files for a cluster named `cluster1` and its two server instances, and saves the archive in a directory named `/space/output`.

```
asadmin> collect-log-files --target cluster1 --retrieve true /space/output
Log files are downloaded for instance1.
Log files are downloaded for instance2.
Created Zip file under /space/output/log_2010-12-15_15-55-54.zip.
Command collect-log-files executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-log-attributes,
- list-log-levels,
- rotate-log,
- set-log-attributes,
- set-log-levels,

- "Administering the Logging Service" in Payara Server General Administration section

Configure-Jms-Cluster

Configures the Message Queue cluster providing JMS services to a Payara Server cluster.

Synopsis

```
asadmin [asadmin-options] configure-jms-cluster [--help]
[--clustertype={conventional|enhanced}]
[--configstoretype={masterbroker|shareddb}]
[--messagestoretype={file|jdbc}]
[--dbvendor database-vendor]
[--dbuser database-user]
[--dburl database-url]
[--property (name=value)[:name=value]*]
cluster-name
```

Description

The `configure-jms-cluster` configures the Message Queue cluster providing JMS services to a Payara Server cluster.

This subcommand should be used before the Payara Server cluster is started for the first time. Otherwise, follow the instructions in administering-the-java-message-service-jms in Payara Server General Administration section.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--clustertype`**: The type of Message Queue cluster to configure. The value `conventional` specifies a conventional cluster, and the value `enhanced` specifies an enhanced, high-availability cluster.
The default value is `conventional`.
If `enhanced` is specified, the `configstoretype` and `messagestoretype` options are ignored. **`--configstoretype`**: The type of data store for configuration data in a

conventional cluster. The value `masterbroker` specifies the use of a master broker to store and manage the configuration data. The value `shareddb` specifies the use of a shared database to store the configuration data.
The default value is `masterbroker`.
This option is ignored if `clustertype` is set to `enhanced`. **`--messagestoretype`**: The type of data store for message data in brokers in a conventional cluster. The value `file` specifies a file store. The value `jdbc` specifies a JDBC store.
The default value is `file`.
This option is ignored if `clustertype` is set to `enhanced`. **`--dbvendor`**: **`--dbuser`**: **`--dburl`**: The database vendor, user, and access url of the JDBC database to use in any of these situations:
* When `clustertype` is set to `enhanced` * When `configstoretype` is set to `shareddb` * When `messagestoretype` is set to `jdbc`

NOTE	To specify the password of the JDBC database user, use the `--passwordfile` utility option of the asadmin-1m command after adding the entry `AS_ADMIN_JMSDBPASSWORD` to the password file.

`--property`: A list of additional database-vendor-specific properties to configure the JDBC database for use by the Message Queue cluster. Specify properties as a colon (`:`) separated list of property names and values in the form:

+

```
prop1name=prop1value:prop2name=prop2value
```

Operands

cluster-name: The name of the Payara Server cluster for which the Message Queue cluster is to provide JMS services.
Because Payara Server uses the cluster name to uniquely identify database tables the Message Queue cluster might require, the length of the name is restricted in the following situations:
* If `clustertype` is set to `enhanced`, the name can be no longer than n−21

characters, where n is the maximum table name length allowed by the database. * If `configstoretype` is set to `shareddb`, the name can be no longer than n–19 characters, where n is the maximum table name length allowed by the database.

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m

Configure-Ldap-For-Admin

Configures the authentication realm named admin-realm for the given LDAP.

Synopsis

```
asadmin [asadmin-options] configure-ldap-for-admin [--help]
```

Description

The `configure-ldap-for-admin` subcommand configures the authentication realm named `admin-realm` for the given LDAP. The `configure-ldap-for-admin` subcommand is interactive. The subcommand prompts the user for the `basedn` and `ldap-group` options.

This command is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 Configuring the LDAP Authentication Realm

```
asadmin> configure-ldap-for-admin
Enter the value for the basedn option>
Enter the value for the ldap-group option>
The LDAP Auth Realm admin-realm was configured correctly
```

```
in admin server's configuration.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- change-admin-password,
- create-auth-realm,
- list-auth-realms

Configure-Managed-Jobs

Configures how long information about subcommands that were started using `asadmin --detach` or that contain progress information is kept.

Synopsis

```
asadmin [asadmin-options] configure-managed-jobs [--help]
[--in-memory-retention-period in-memory-retention-period]
[--job-retention-period job-retention-period]
[--cleanup-initial-delay cleanup-initial-delay]
[--cleanup-poll-interval cleanup-poll-interval]
```

Description

The `configure-managed-jobs` subcommand configures how long information about subcommands (jobs) that were started using the `asadmin` utility option `--detach` or that contain progress information is kept. The `--detach` option detaches long-running subcommands and executes them in the background in detach mode. Job information includes subcommand progress and status.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--in-memory-retention-period`**: Specifies how long a completed

job is kept in memory after the job is finished. The default value is 1 hour. `--job-retention-period`: Specifies how long a job is stored. The default value is 24 hours. `--cleanup-initial-delay`: After server startup, specifies the initial delay after which the cleanup service starts purging jobs. The default value is 5 minutes. `--cleanup-poll-interval`: Specifies the time interval after which the cleanup service polls for expired jobs. The default value is 20 minutes.

Examples

Example 1 Configuring the Job Retention Period

This example sets the job retention period to 36 hours. Time periods can be specified in Hh|Mm|Ss for hours, minutes, or seconds.

```
asadmin> configure-managed-jobs --job-retention-period=36h
Command configure-managed-jobs executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-jobs

Copy-Config

Copies an existing named configuration to create another configuration.

Synopsis

```
asadmin [asadmin-options] copy-config [--help]
[--systemproperties (name=value)[:name=value]*]
source-configuration-name destination-configuration-name
```

Description

The `copy-config` subcommand creates a named configuration in the configuration of the domain administration server (DAS) by copying an existing configuration. The new configuration is identical to the copied configuration, except for any properties that you

specify in the `--systemproperties` option.

The `default-config` configuration is copied when a standalone sever instance or standalone cluster is created.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--systemproperties`**: Optional attribute name-value pairs for the configuration. These properties override port settings in the configuration.
The following properties are available:
`ASADMIN_LISTENER_PORT`;; This property specifies the port number of the HTTP port or HTTPS port through which the DAS connects to the instance to manage the instance. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `HTTP_LISTENER_PORT`;; This property specifies the port number of the port that is used to listen for HTTP requests. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `HTTP_SSL_LISTENER_PORT`;; This property specifies the port number of the port that is used to listen for HTTPS requests. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `IIOP_LISTENER_PORT`;; This property specifies the port number of the port that is used for IIOP connections. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `IIOP_SSL_LISTENER_PORT`;; This property specifies the port number of the port that is used for secure IIOP connections. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `IIOP_SSL_MUTUALAUTH_PORT`;; This property specifies the port number of the port that is used for secure IIOP connections with client authentication. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `JAVA_DEBUGGER_PORT`;; This property specifies the port number of the port that is used for connections to the Java Platform

Debugger Architecture (JPDA) (http://www.oracle.com/technetwork/java/javase/tech/jpda-141715.html) debugger. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `JMS_PROVIDER_PORT`;; This property specifies the port number for the Java Message Service provider. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `JMX_SYSTEM_CONNECTOR_PORT`;; This property specifies the port number on which the JMX connector listens. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `OSGI_SHELL_TELNET_PORT`;; This property specifies the port number of the port that is used for connections to the Apache Felix Remote Shell (`http://felix.apache.org/site/apache-felix-remote-shell.html`). This shell uses the Felix shell service to interact with the OSGi module management subsystem. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges.

Operands

source-configuration-name: The name of the configuration that you are copying.
destination-configuration-name: The name of the configuration that you are creating by copying the source configuration.
The name must meet the following requirements:
* The name may contain only ASCII characters. * The name must start with a letter, a number, or an underscore. * The name may contain only the following characters: **Lowercase letters** Uppercase letters **Numbers** Hyphen **Period** Underscore * The name must be unique in the domain and must not be the name of a another named configuration, a cluster, a Payara Server instance, or a node. * The name must not be `domain`, `server`, or any other keyword that is reserved by Payara Server.

Examples

Example 1 Copying a Configuration

This example copies the `default-config` configuration to the `pmdsaconfig` configuration, overriding the settings for the following ports:

- HTTP listener port
- HTTPS listener port

```
asadmin> copy-config
--systemproperties HTTP_LISTENER_PORT=2000:HTTP_SSL_LISTENER_PORT=3000
default-config pmdsaconfig

Command copy-config executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- delete-config,
- list-configs

Create-Admin-Object

Adds the administered object with the specified JNDI name for a resource adapter.

Synopsis

```
asadmin [asadmin-options] create-admin-object [--help]
[--target target]
--restype restype
[--classname classname]
--raname raname
[--enabled={true|false}]
[--description description]
[--property name=value[:name=value]*]
jndi_name
```

Description

The `create-admin-object` subcommand creates the administered object with the specified JNDI name and the interface definition for a resource adapter.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options,

see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--target`: Specifies the target on which you are creating the administered object. Valid values are as follows:
`server`;; Creates the administered object for the default server instance `server` and is the default value. configuration_name;; Creates the administered object for the named configuration. cluster_name;; Creates the administered object for every server instance in the cluster. instance_name;; Creates the administered object for a particular server instance.

NOTE	The resource is always created for the domain as a whole, but the `resource-ref` for the resource is only created for the specified `--target`. This means that although the resource is defined at the domain level, it is only available at the specified target level. Use the `create-resource-ref` subcommand to refer to the resource in multiple targets if needed.

`--restype`: Specifies the interface definition for the administered object. The resource type must be an interface definition that is specified in the `ra.xml` file of the resource adapter. `--classname`: Specifies the class name of the administered object. Required if multiple administered objects use the same interface definition. `--raname`: Specifies the name of the resource adapter associated with this administered object. `--enabled`: Specifies if this object is enabled. Default is true. `--description`: Text string describing the administered object. `--property`: Description of the name/values pairs for configuring the resource. Dependent on the resource adapter. For JMS properties, see create-jms-resource for JMS destination resources.

Operands

jndi_name: JNDI name of the administered object to be created.

Examples

Example 1 Creating an Administered Object

In this example, `jmsra` is a system resource adapter with the admin object interfaces,

`jakarta.jms.Queue` and `jakarta.jms.Topic`.

```
asadmin> create-admin-object --restype jakarta.jms.Queue
--raname jmsra --description "sample administered object"
--property Name=sample_jmsqueue jms/samplequeue
Command create-admin-object executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-resource-ref,
- delete-admin-object,
- list-admin-objects

Create-Application-Ref

Creates a reference to an application.

Synopsis

```
asadmin [asadmin-options] create-application-ref [--help]
[--target target]
[--virtualservers virtual_servers] [--enabled=true]
[--lbenabled=true] reference_name
```

Description

The `create-application-ref` subcommand creates a reference from a cluster or an unclustered server instance to a previously deployed application element (for example, a Java EE application, a Web module, or an enterprise bean module). This effectively results in the application element being deployed and made available on the targeted instance or cluster.

The target instance or instances making up the cluster need not be running or available for this subcommand to succeed. If one or more instances are not available, they will receive the new application element the next time they start.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which you are creating the application reference. Valid values are
* `server` - Specifies the default server instance as the target for creating the application reference. `server` is the name of the default server instance and is the default value for this option. * cluster_name - Specifies a particular cluster as the target for creating the application reference. * instance_name - Specifies a particular stand-alone server instance as the target for creating the application reference.
`--virtualservers`: Specifies a comma-separated list of virtual server IDs on which to deploy. This option applies only to Web modules (either standalone or in a Java EE application). If this option is not specified, the application is deployed to all virtual servers except the administrative server, `__asadmin`. **`--enabled`**: Indicates whether the application should be enabled (that is, loaded). This value will take effect only if the application is enabled at the global level. The default is `true`. **`--lbenabled`**: Controls whether the deployed application is available for load balancing. The default is true.

Operands

reference_name: The name of the application or module, which can be a Java EE application, Web module, EJB module, connector module or application client module. The name can include an optional version identifier, which follows the name and is separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. If the `--enabled` option is set to false, you can create references to multiple disabled versions by using an asterisk (`*`) as a wildcard character. For more information about module and application versions, see "module-and-application-versions" in Payara Server Application Deployment section.

Examples

Example 1 Creating an Application Reference

The following example creates a reference to the Web module `MyWebApp` on the unclustered server instance `NewServer`.

```
asadmin> create-application-ref --target NewServer MyWebApp
Command create-application-ref executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- delete-application-ref,
- list-application-refs
- "Payara Server Application Deployment"

Create-Auth-Realm

Adds the named authentication realm.

Synopsis

```
asadmin [asadmin-options] create-auth-realm [--help]
--classname realm_class [--property(name=value)[:name=value]*]
[--target target_name] auth_realm_name
```

Description

The `create-auth-realm` subcommand adds the named authentication realm.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target on which you are creating the realm. Valid values are

`server`;; Creates the realm on the default server instance. This is the default value.

configuration_name;; Creates the realm in the specified configuration. cluster_name;; Creates the realm on all server instances in the specified cluster. instance_name;; Creates the realm on a specified server instance. **`--classname`**: Java class which implements this realm. These include `com.sun.enterprise.security.auth.realm.file.FileRealm`, `com.sun.enterprise.security.auth.realm.certificate.CertificateRealm`, `com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm`, `com.sun.enterprise.security.auth.realm.ldap.LDAPRealm`, `com.sun.enterprise.security.auth.realm.ldap.PamRealm`, and `com.sun.enterprise.security.auth.realm.solaris.SolarisRealm`, or a custom realm. **`--property`**: Optional attribute name-value pairs for configuring the authentication realm. Authentication realms require provider-specific properties, which vary based on implementation.

The following properties are common to all of the supported realms, which include `FileRealm`, `CertificateRealm`, `JDBCRealm`, `LDAPRealm`, PamRealm, and `SolarisRealm`.

`jaas-context`;; Specifies the Java Authentication and Authorization Service (JAAS) context. `assign-groups`;; (Optional) If this property is set, its value is taken to be a comma-separated list of group names. All clients who present valid certificates are assigned membership to these groups for the purposes of authorization decisions in the web and EJB containers.

Specific to each realm, you can specify the following properties.

* You can specify the following properties for `FileRealm`:

`file`;; Specifies the file that stores usernames, passwords, and group names. The default is domain-dir`/config/keyfile`. * You can specify the following properties for `CertificateRealm`:

`LoginModule`;; Specifies the name of a JAAS `LoginModule` to use for performing authentication. To use a JAAS `LoginModule`, you must first create an implementation of the javax.security.auth.spi.LoginModule interface, and then plug the module into a `jaas-context`. * You can specify the following properties for `JDBCRealm`:

`datasource-jndi`;; Specifies the `jndi-name` of the `jdbc-resource` for the

database. `user-table`;; Specifies the name of the user table in the database. `user-name-column`;; Specifies the name of the user name column in the database's user table. `password-column`;; Specifies the name of the password column in the database's user table. `group-table`;; Specifies the name of the group table in the database. `group-table`;; Specify the group table for an authentication realm of class `JDBCRealm`. `group-name-column`;; Specifies the name of the group name column in the database's group table. `db-user`;; (Optional) Allows you to specify the database username in the realm instead of the `jdbc-connection-pool`. This prevents other applications from looking up the database, getting a connection, and browsing the user table. By default, the `jdbc-connection-pool` configuration is used. `db-password`;; (Optional) Allows you to specify the database password in the realm instead of the `jdbc-connection-pool`. This prevents other applications from looking up the database, getting a connection, and browsing the user table. By default, the `jdbc-connection-pool` configuration is used. `group-table`;; Specifies the name of the group table in the database. `digest-algorithm`;; (Optional) Specifies the digest algorithm. The default is `SHA-256`. You can use any algorithm supported in the JDK, or none.

NOTE

If you have applications that depend on the `MD5` algorithm, you can override the default `SHA-25` algorithm by using the `asadmin set` subcommand:

```
asadmin> set server.security-service.property.default-digest-
algorithm=MD5
```

You can use the `asadmin get` subcommand to determine what algorithm is currently being used:

```
asadmin> get server.security-service.property.default-digest-
algorithm
```

`digestrealm-password-enc-algorithm`

(Optional) Specifies the algorithm for encrypting passwords stored in the

database.

NOTE	It is a security risk not to specify a password encryption algorithm.

`encoding`

(Optional) Specifies the encoding. Allowed values are `Hex` and `Base64`. If digest-algorithm is specified, the default is `Hex`. If `digest-algorithm` is not specified, by default no encoding is specified.

`charset`

(Optional) Specifies the `charset` for the digest algorithm.

- You can specify the following properties for `LDAPRealm`:

`directory`

Specifies the LDAP URL to your server.

`base-dn`

Specifies the LDAP base DN for the location of user data. This base DN can be at any level above the user data, since a tree scope search is performed. The smaller the search tree, the better the performance.

`search-filter`

(Optional) Specifies the search filter to use to find the user. The default is `uid=%s` (`%s` expands to the subject name).

`group-base-dn`

(Optional) Specifies the base DN for the location of groups data. By default, it is same as the `base-dn`, but it can be tuned, if necessary.

`group-search-filter`

(Optional) Specifies the search filter to find group memberships for the user. The default is `uniquemember=%d` (`%d` expands to the user `elementDN`).

`group-target`

(Optional) Specifies the LDAP attribute name that contains group name entries. The default is `CN`.

`search-bind-dn`

(Optional) Specifies an optional DN used to authenticate to the directory for performing the search-filter lookup. Only required for directories that do not allow anonymous search.

`search-bind-password`

(Optional) Specifies the LDAP password for the DN given in `search-bind-dn`.

Operands

auth_realm_name: A short name for the realm. This name is used to refer to the realm from, for example, `web.xml`.

Examples

Example 1 Creating a New Authentication Realm

This example creates a new file realm.

```
asadmin> create-auth-realm
--classname com.sun.enterprise.security.auth.realm.file.FileRealm
--property file=${com.sun.aas.instanceRoot}/config/admin-keyfile:jaas-context
=fileRealm file
Command create-auth-realm executed successfully
```

Where `file` is the authentication realm created.

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-auth-realm,

- list-auth-realms

Create-Cluster

Creates a Payara Server cluster.

Synopsis

```
asadmin [asadmin-options] create-cluster [--help]
[--config config-name]
[--systemproperties (name=value)[:name=value]*]
[--properties (name=value)[:name=value]*]
[--gmsenabled={true|false}]
[--multicastport multicast-port]
[--multicastaddress multicast-address]
[--bindaddress bind-address]
cluster-name
```

Description

The `create-cluster` subcommand creates a Payara Server cluster. Initially the cluster contains no Payara Server instances, applications, or resources.

A cluster requires a reference to the named configuration that defines the configuration of all instances that are added to the cluster. The configuration can be specified in the command to create the cluster, but is not required. If no configuration is specified, the subcommand creates a configuration that is named cluster-name`-config` for the cluster. The cluster that is created is a standalone cluster because the cluster's configuration is not shared with any other clusters or standalone instances.

To add instances to the cluster, set the `--cluster` option to the name of the cluster when using either of the following subcommands:

- create-instance
- create-local-instance

To delete server instances from the cluster at any time, use one of the following subcommands:

- delete-instance

- delete-local-instance

To associate applications and resources with all instances in the cluster, set the `--target` option to the name of the cluster when performing the following operations:

- Deploying applications by using the deploy subcommand
- Creating resources by using subcommands such as create-jdbc-resource
- Creating references to applications that are already deployed in other targets by using the create-application-ref subcommand
- Creating references to resources that are already created in other targets by using the create-resource-ref subcommand

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--config`: Specifies the named configuration that the cluster references. The configuration must exist and must not be named `default-config` or `server-config`. Specifying the `--config` option creates a shared cluster. If this option is omitted, a standalone cluster is created.

`--systemproperties`: Defines system properties for the configuration that is created for the cluster. These properties override the property values in the `default-config` configuration. The following properties are available:
`ASADMIN_LISTENER_PORT`;; This property specifies the port number of the HTTP port or HTTPS port through which the DAS connects to the instance to manage the instance. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `HTTP_LISTENER_PORT`;; This property specifies the port number of the port that is used to listen for HTTP requests. Valid values are 1-65535. On UNIX,

creating sockets that listen on ports 1-1024 requires superuser privileges. `HTTP_SSL_LISTENER_PORT`;; This property specifies the port number of the port that is used to listen for HTTPS requests. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `IIOP_LISTENER_PORT`;; This property specifies the port number of the port that is used for IIOP connections. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `IIOP_SSL_LISTENER_PORT`;; This property specifies the port number of the port that is used for secure IIOP connections. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `IIOP_SSL_MUTUALAUTH_PORT`;; This property specifies the port number of the port that is used for secure IIOP connections with client authentication. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `JAVA_DEBUGGER_PORT`;; This property specifies the port number of the port that is used for connections to the Java Platform Debugger Architecture (JPDA) (http://www.oracle.com/technetwork/java/javase/tech/jpda-141715.adoc) debugger. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `JMS_PROVIDER_PORT`;; This property specifies the port number for the Java Message Service provider. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `JMX_SYSTEM_CONNECTOR_PORT`;; This property specifies the port number on which the JMX connector listens. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `OSGI_SHELL_TELNET_PORT`;; This property specifies the port number of the port that is used for connections to the Apache Felix Remote Shell (http://felix.apache.org/site/apache-felix-remote-shell.html). This shell uses the Felix shell service to interact with the OSGi module management subsystem. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. **`--properties`**: Defines properties for the cluster. The following properties are available:

`GMS_DISCOVERY_URI_LIST`

The locations of Payara Server instances in the cluster to use for discovering the

cluster. This property is required only if the Group Management Service (GMS) is not using multicast for broadcasting messages.

Valid values for this property are as follows:

- A comma-separated list of uniform resource identifiers (URIs). Each URI must locate a Payara Server instance or the DAS. This format is required if multiple Payara Server instances are running on the same host.

  The format of each URI in the list is as follows:

  scheme `://` host-name-or -IP-address `:` port

  - scheme is the URI scheme, which is `tcp`.
  - host-name-or -IP-address is the host name or IP address of the host on which the instance is running.
  - port is the port number of the port on which the instance listens for messages from GMS. The system property `GMS_LISTENER_PORT-` clustername must be set for the instance.
- A comma-separated list of IP addresses or host names on which the DAS or the instances are running. The list can contain a mixture of IP addresses and host names. This format can be used only if one clustered instance is running on each host. The value of the `GMS_LISTENER_PORT` property must be unique for each cluster in a domain.
- The keyword `generate`. This format can be used only if one instance in a cluster is running on each host and the DAS is running on a separate host. Multiple instances on the same host cannot be members of the same cluster. The value of the `GMS_LISTENER_PORT` property must be unique for each cluster in a domain.

`GMS_LISTENER_PORT`

The port number of the port on which the cluster listens for messages from GMS.

The default value is a reference to the `GMS_LISTENER_PORT-` cluster-name system property. By default, this system property is not set. In this situation, GMS selects a free port from the range that is defined by the properties `GMS_TCPSTARTPORT` and `GMS_TCPENDPORT`. By default, this range is 9090-9200. In most situations, the default behavior should suffice.

However, if GMS is not using multicast for broadcasting messages, the `GMS_LISTENER_PORT` property must specify a port number that is valid for all Payara Server instances in the cluster. To use the default value to meet this requirement, use a system property to set the port number individually for each instance.

For example, use the `create-system-properties` subcommand to create the system property `GMS_LISTENER_PORT-` cluster-name for the DAS. Then, for each instance in the cluster, set the `GMS_LISTENER_PORT-` cluster-name system property to the port number on which the instance listens for messages from GMS. The default value of the `GMS_LISTENER_PORT` property for the cluster references this system property.

`GMS_LOOPBACK`

Specifies whether an instance may receive from itself application-level messages that the instance broadcasts to the cluster.

Possible values are as follows:

`false`

The instance may not receive messages from itself (default).

`true`

The instance may receive messages from itself. Use this setting for testing an instance when the instance is the only instance in a cluster.

GMS_MULTICAST_TIME_TO_LIVE

The maximum number of iterations or transmissions that a multicast message for the following types of events can experience before the message is discarded:

- Group discovery
- Member heartbeats
- Membership changes

To match the configuration of the network on which the DAS and clustered instances are deployed, set this value as low as possible. To determine the lowest possible value for your system, use the validate-multicast subcommand.

A value of 0 ensures that multicast messages never leave the host from which they are broadcast.

A value of 1 might prevent the broadcast of messages between hosts on same subnet that are connected by a switch or a router.

The default is 4, which ensures that messages are successfully broadcast to all cluster members in networks where hosts are connected by switches or routers.

GMS_TCPENDPORT

The highest port number in the range from which GMS selects a free port if the `GMS_LISTENER_PORT-`cluster-name system property is not set. The default is 9200.

GMS_TCPSTARTPORT

The lowest port number in the range from which GMS selects a free port if the `GMS_LISTENER_PORT-`cluster-name system property is not set. The default is 9090.--gmsenabled`: Specifies whether GMS is enabled for the cluster.

Possible values are as follows:

`true`

GMS is enabled for the cluster (default).

When GMS is enabled for a cluster, GMS is started in each server instance in the cluster and in the DAS. The DAS participates in each cluster for which this option is set to `true`.

`false`

GMS is disabled for the cluster.

`--multicastaddress`: The address on which GMS listens for group events. This option must specify a multicast address in the range 224.0.0.0 through 239.255.255.255. The default is 228.9.XX.YY, where XX and YY are automatically generated independent values between 0 and 255.

`--multicastport`: The port number of communication port on which GMS listens for group events. This option must specify a valid port number in the range 2048-49151. The default is an automatically generated value in this range.

`--bindaddress`: The Internet Protocol (IP) address of the network interface to which GMS binds. This option must specify the IP address of a local network interface. The default is all public network interface addresses.

+ On a multihome machine, this option configures the network interface that is used for the GMS. A multihome machine possesses two or more network interfaces.

+ To specify an address that is valid for all Payara Server instances in the cluster, use a system property to set the address individually for each instance.

+ For example, use the `create-system-properties` subcommand to create the system property `GMS-BIND-INTERFACE-ADDRESS-` cluster-name. Then set the `--bindaddress` option of this subcommand to `${GMS-BIND-INTERFACE-ADDRESS-` cluster-name `}` to specify the system property. Finally, for each instance in the cluster, set the `GMS-BIND-INTERFACE-ADDRESS-` cluster-name system property

to the required network interface address on the instance's machine.

Operands

cluster-name: The name of the cluster.

+ The name must meet the following requirements:

+ * The name may contain only ASCII characters. * The name must start with a letter, a number, or an underscore. * The name may contain only the following characters: **Lowercase letters** Uppercase letters **Numbers** Hyphen **Period** Underscore * The name must be unique in the domain and must not be the name of another cluster, a named configuration, a Payara Server instance, or a node. * The name must not be `domain`, `server`, or any other keyword that is reserved by Payara Server.
If the configure-jms-cluster subcommand is to be used to configure a Message Queue cluster to provide JMS services to the Payara Server cluster, the length of the Payara Server cluster name is might be restricted:
* If `clustertype` is set to `enhanced` in the configure-jms-cluster subcommand, the name can be no longer than n–21 characters, where n is the maximum table name length allowed by the database. * If `configstoretype` is set to `shareddb` in the configure-jms-cluster subcommand, the name can be no longer than n–19 characters, where n is the maximum table name length allowed by the database.

Examples

Example 1 Creating a Cluster

This example creates a cluster that is named `ltscluster` for which port 1169 is to be used for secure IIOP connections. Because the `--config` option is not specified, the cluster references a copy of the named configuration `default-config` that is named `ltscluster-config`.

```
asadmin> create-cluster
--systemproperties IIOP_SSL_LISTENER_PORT=1169
ltscluster
Command create-cluster executed successfully.
```

Example 2 Creating a Cluster With a List of URIs for Discovering the Cluster

This example creates a cluster that is named `tcpcluster`. In this example, GMS is not using multicast for broadcasting messages and multiple instances reside on the same host. Therefore, the `GMS_DISCOVERY_URI_LIST` property is set to the locations of the Payara Server instances to use for discovering the cluster. These instances reside on the host whose IP address is `10.152.23.224` and listen for GMS events on ports 9090, 9091, and 9092.

To distinguish colon (`:`) characters in URIs from separators in a property list, colons in URIs are escaped with single quote characters (`'`) and backslash (`\`) characters. For more information about escape characters in options for the `asadmin` utility, see the asadmin-1m help page.

This example assumes that the port on which each instance listens for GMS messages is set independently for the instance through the `GMS_LISTENER_PORT-tcpcluster` system property.

```
asadmin> create-cluster --properties GMS_DISCOVERY_URI_LIST=
tcp'\\:'//10.152.23.224'\\:'9090,
tcp'\\:'//10.152.23.224'\\:'9091,
tcp'\\:'//10.152.23.224'\\:'9092 tcpcluster
Command create-cluster executed successfully.
```

Example 3 Creating a Cluster With a List of IP Addresses for Discovering the Cluster

This example creates a cluster that is named `ipcluster`. In this example, GMS is not using multicast for broadcasting messages and only one clustered instance resides on each host. Therefore, the `GMS_DISCOVERY_URI_LIST` property is set to the IP addresses of the hosts where instances to use for discovering the cluster are running. The cluster listens for messages from GMS on port 9090.

```
asadmin> create-cluster --properties 'GMS_DISCOVERY_URI_LIST=
10.152.23.225,10.152.23.226,10.152.23.227,10.152.23.228:
GMS_LISTENER_PORT=9090' ipcluster
Command create-cluster executed successfully.
```

Example 4 Creating a Cluster With a Generated List of Instances for Discovering the Cluster

This example creates a cluster that is named `gencluster`. In this example, GMS is not using multicast for broadcasting messages, one instance in the cluster is running on each host and the DAS is running on a separate host. Therefore, the `GMS_DISCOVERY_URI_LIST` property is set to the keyword `generate` to generate a list of instances to use for discovering the cluster. The cluster listens for messages from GMS on port 9090.

```
asadmin> create-cluster --properties 'GMS_DISCOVERY_URI_LIST=generate:
GMS_LISTENER_PORT=9090' gencluster
Command create-cluster executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-application-ref,
- create-instance,
- create-jdbc-resource,
- create-local-instance,
- create-resource-ref,
- delete-cluster,
- delete-instance,
- delete-local-instance,
- deploy,
- list-clusters,
- start-cluster,

- stop-cluster,
- restart-cluster,
- validate-multicast
- Apache Felix Remote Shell (http://felix.apache.org/site/apache-felix-remote-shell.html), Java
- http://www.oracle.com/technetwork/java/javase/tech/jpda-141715.htmlPlatform Debugger Architecture (JPDA) (http://www.oracle.com/technetwork/java/javase/tech/jpda-141715.html)

Create-Connector-Connection-Pool

Adds a connection pool with the specified connection pool name.

Synopsis

```
asadmin [asadmin-options] create-connector-connection-pool [--help]
--raname raname
--connectiondefinition connectiondefinitionname
[--steadypoolsize steadypoolsize]
[--maxpoolsize maxpoolsize]
[--maxwait maxwait]
[--poolresize poolresize]
[--idletimeout idletimeout]
[--isconnectvalidatereq={false|true}]
[--failconnection={false|true}]
[--leaktimeout=timeout]
[--leakreclaim={false|true}]
[--creationretryattempts=attempts]
[--creationretryinterval=interval]
[--lazyconnectionenlistment={false|true}]
[--lazyconnectionassociation={false|true}]
[--associatewiththread={false|true}]
[--matchconnections={true|false}]
[--maxconnectionusagecount=count]
[--validateatmostonceperiod=interval]
[--transactionsupport transactionsupport]
[--descrip[tion description]
[--ping {false|true}]
[--pooling {true|false}]
[--property (name=value)[:name=value]*]
poolname
```

Description

The `create-connector-connection-pool` subcommand defines a pool of connections to an enterprise information system (EIS). The named pool can be referred

to by multiple connector resources. Each defined pool is instantiated at server startup, and is populated when accessed for the first time. If two or more connector resources point to the same connector connection pool, they are using the same pool of connections at run time. There can be more than one pool for a connection definition in a single resource adapter.

A connector connection pool with authentication can be created either by using a `--property` option to specify user, password, or other connection information, or by specifying the connection information in the XML descriptor file.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--associatewiththread`**: Specifies whether a connection is associated with the thread to enable the thread to reuse the connection. If a connection is not associated with the thread, the thread must obtain a connection from the pool each time that the thread requires a connection. Possible values are as follows: `false`;; A connection is not associated with the thread (default). `true`;; A connection is associated with the thread. **`--connectiondefinition`**: The name of the connection definition. **`--creationretryattempts`**: Specifies the maximum number of times that the server retries to create a connection if the initial attempt fails.
Default value is 0, which specifies that the server does not retry to create the connection. **`--creationretryinterval`**: Specifies the interval, in seconds, between successive attempts to create a connection.
If `--creationretryattempts` is 0, the `--creationretryinterval` option is ignored. Default value is 10. **`--description`**: Text providing descriptive details about the connector connection pool. **`--failconnection`**: If set to true, all connections in the pool are closed if a single validation check fails. This parameter is mandatory if the `--isconnectvalidatereq` option is set to true. Default value is false.
`--idletimeout`: The maximum time that a connection can remain idle in the pool.

After this amount of time, the pool can close this connection. Default value is 300.
`--isconnectvalidatereq`: If the value is set to true, the connections will be checked to see if they are usable, before they are given out to the application. Default value is false. `--lazyconnectionenlistment`: Specifies whether a resource to a transaction is enlisted only when a method actually uses the resource. Default value is false.
`--lazyconnectionassociation`: Specifies whether a physical connection should be associated with the logical connection only when the physical connection is used, and disassociated when the transaction is completed. Such association and dissociation enable the reuse of physical connections. Possible values are as follows:
`false`;; A physical connection is associated with the logical connection even before the physical connection is used, and is not disassociated when the transaction is completed (default). `true`;; A physical connection is associated with the logical connection only when the physical connection is used, and disassociated when the transaction is completed. The `--lazyconnectionenlistment` option must also be set to `true`. `--leakreclaim`: Specifies whether leaked connections are restored to the connection pool after leak connection tracing is complete. Possible values are as follows:
`false`;; Leaked connections are not restored to the connection pool (default). `true`;; Leaked connections are restored to the connection pool. `--leaktimeout`: Specifies the amount of time, in seconds, for which connection leaks in a connection pool are to be traced.
If connection leak tracing is enabled, you can use the Administration Console to enable monitoring of the JDBC connection pool to get statistics on the number of connection leaks. Default value is 0, which disables connection leak tracing.
`--matchconnections`: Specifies whether a connection that is selected from the pool should be matched with the resource adaptor. If all connections in the pool are identical, matching between connections and resource adapters is not required. Possible values are as follows:
`true`;; A connection should be matched with the resource adaptor (default). `false`;; A connection should not be matched with the resource adaptor.
`--maxconnectionusagecount`: Specifies the maximum number of times that a

connection can be reused.
When this limit is reached, the connection is closed. Default value is 0, which specifies no limit on the number of times that a connection can be reused. **`--maxpoolsize`**: The maximum number of connections that can be created to satisfy client requests. Default value is 32. **`--maxwait`**: The amount of time, in milliseconds, that a caller must wait before a connection is created, if a connection is not available. If set to 0, the caller is blocked indefinitely until a resource is available or until an error occurs. Default value is 60000. **`--ping`**: A pool with this attribute set to true is contacted during creation (or reconfiguration) to identify and warn of any erroneous values for its attributes. Default value is false. **`--pooling`**: When set to false, this attribute disables connection pooling. Default value is true. **`--poolresize`**: Quantity by which the pool will scale up or scale down the number of connections. Scale up: When the pool has no free connections, pool will scale up by this quantity. Scale down: All the invalid and idle connections are removed, sometimes resulting in removing connections of quantity greater than this value. The number of connections that is specified by `--steadypoolsize` will be ensured. Possible values are from 0 to `MAX_INTEGER`. Default value is 2. **`--property`**: Optional attribute name/value pairs for configuring the pool. **`--raname`**: The name of the resource adapter. **`--steadypoolsize`**: The minimum and initial number of connections maintained in the pool. Default value is 8. **`--transactionsupport`**: Indicates the level of transaction support that this pool will have. Possible values are `XATransaction`, `LocalTransaction` and `NoTransaction`. This attribute can have a value lower than or equal to but not higher than the resource adapter's transaction support attribute. The resource adapter's transaction support attribute has an order of values, where `XATransaction` is the highest, and `NoTransaction` the lowest. **`--validateatmostonceperiod`**: Specifies the time interval in seconds between successive requests to validate a connection at most once. Setting this attribute to an appropriate value minimizes the number of validation requests by a connection. Default value is 0, which means that the attribute is not enabled.

Operands

poolname: The name of the connection pool to be created.

Examples

Example 1 Creating a Connector Connection Pool

This example creates a new connector connection pool named `jms/qConnPool`.

```
asadmin> create-connector-connection-pool --raname jmsra
--connectiondefinition jakarta.jms.QueueConnectionFactory --steadypoolsize 20
--maxpoolsize 100 --poolresize 2 --maxwait 60000 jms/qConnPool
Command create-connector-connection-pool executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- delete-connector-connection-pool,
- list-connector-connection-pools,
- ping-connection-pool

Create-Connector-Resource

Registers the connector resource with the specified JNDI name.

Synopsis

```
asadmin [asadmin-options] create-connector-resource [--help]
--poolname connectorConnectionPoolName
[--enabled={true|false}]
[--description description]
[--objecttype ovjecttype]
[--property (name=value)[:name=value]*]
[--target target]
jndi_name
```

Description

The `create-connector-resource` subcommand registers the connector resource with the specified JNDI name.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--poolname`**: The name of the connection pool. When two or more resource elements point to the same connection pool element, they use the same pool connections at runtime. **`--enabled`**: This option determines whether the resource is enabled at runtime. The default value is true. **`--objecttype`**: Defines the type of the connector resource. Default is user. Allowed values are:
`system-all`;; A system resource for all server instances and the domain administration server (DAS). `system-admin`;; A system resource only for the DAS. `system-instance`;; A system resource for all server instances only. `user`;; A user resource. **`--description`**: Text providing details about the connector resource. **`--property`**: Optional attribute name value pairs for configuring the resource. **`--target`**: This option specifies the ending location of the connector resources. Valid targets are:
`server`;; Creates the connector resource in the default server instance. This is the default value. `domain`;; Creates the connector resource in the domain. cluster_name;; Creates the connector resource in every server instance in the cluster. instance_name;; Creates the connector resource in the specified server instance.

NOTE	The resource is always created for the domain as a whole, but the `resource-ref` for the resource is only created for the specified `--target`. This means that although the resource is defined at the domain level, it is only available at the specified target level. Use the `create-resource-ref` subcommand to refer to the resource in multiple targets if needed.

Operands

jndi_name: The JNDI name of this connector resource.

Examples

Example 1 Creating a Connector Resource

This example creates a connector resource named `jms/qConnFactory`.

```
asadmin> create-connector-resource --poolname jms/qConnPool
--description "sample connector resource" jms/qConnFactory
Command create-connector-resource executed successfully
```

Example 2 Using the create-connector-resource subcommand

This example shows the usage of this subcommand.

```
asadmin> create-connector-resource --target server --poolname jms/qConnPool
--description "sample connector resource" jms/qConnFactory
Command create-connector-resource executed successfully
```

Where `jms/qConnFactory` is the sample connector resource that is created.

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- delete-connector-resource,
- list-connector-resources,
- create-resource-ref

Create-Connector-Security-Map

Creates a security map for the specified connector connection pool.

Synopsis

```
asadmin [asadmin-options] create-connector-security-map [--help]
--poolname connector_connection_pool_name
[--principals principal-name1[,principal-name2]*]
[--usergroups user-group1[,user-group2]*]
[--mappedusername user-name]
mapname
```

Description

The `create-connector-security-map` subcommand creates a security map for

the specified connector connection pool. If the security map is not present, a new one is created. This subcommand can also map the caller identity of the application (principal or user group) to a suitable enterprise information system (EIS) principal in container-managed authentication scenarios. The EIS is any system that holds the data of an organization. It can be a mainframe, a messaging system, a database system, or an application. One or more named security maps can be associated with a connector connection pool. The connector security map configuration supports the use of the wild card asterisk (`*`) to indicate all users or all user groups.

To specify the EIS password, you can add the `AS_ADMIN_MAPPEDPASSWORD` entry to the password file, then specify the file by using the `--passwordfile asadmin` utility option.

For this subcommand to succeed, you must have first created a connector connection pool using the `create-connector-connection-pool` subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--poolname`**: Specifies the name of the connector connection pool to which the security map belongs. **`--principals`**: Specifies a list of backend EIS principals. More than one principal can be specified using a comma-separated list. Use either the `--principals` or `--usergroups` options, but not both in the same command. **`--usergroups`**: Specifies a list of backend EIS user group. More than one user groups can be specified using a comma separated list. Use either the `--principals` or `--usergroups` options, but not both in the same command. **`--mappedusername`**: Specifies the EIS username.

Operands

mapname: The name of the security map to be created.

Examples

Example 1 Creating a Connector Security Map

This example creates `securityMap1` for the existing connection pool named `connector-pool1`.

```
asadmin> create-connector-security-map --poolname connector-pool1
--principals principal1,principal2 --mappedusername backend-username securityMap1
Command create-connector-security-map executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- delete-connector-security-map,
- list-connector-security-maps,
- update-connector-security-map

Create-Connector-Work-Security-Map

Creates a work security map for the specified resource adapter.

Synopsis

```
asadmin [asadmin-options] create-connector-work-security-map [--help]
--raname raname
[--principalsmap eis-principal1=principal_name1[, eis-principal2=principal_name2]*
|--groupsmap eis-group1=server-group1[, eis-group2=server-group2]*}
[--description description]
mapname
```

Description

The `create-connector-work-security-map` subcommand maps the caller identity of the work submitted by the resource adapter EIS principal or EIS user group to a suitable principal or user group in the Payara Server security domain. One or more work security maps may be associated with a resource adapter. The connector work security map configuration supports the use of the wild card asterisk (`*`) to indicate all

users or all user groups.

The enterprise information system (EIS) is any system that holds the data of an organization. It can be a mainframe, a messaging system, a database system, or an application.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--description`**: Text providing descriptive details about the connector work security map. **`--groupsmap`**: Specifies a map of the backend EIS user group to the Payara Server user group. Use a comma-separated list to specify more than one mapping. Use either the `--principalsmap` option or the `--groupsmap` option, but not both. **`--principalsmap`**: Specifies a map of the backend EIS principal to the Payara Server principal. Use a comma-separated list to specify more than one mapping. Use either the `--principalsmap` option or the `--groupsmap` option, but not both. **`--raname`**: Indicates the connector module name, which is the name of the resource adapter.

Operands

mapname: The name of the work security map to be created.

Examples

Example 1 Creating a Connector Work Security Map (Principal)

This example creates connector work security map `workSecurityMap1` that maps the backend EIS principal to the Payara Server principal.

```
asadmin create-connector-work-security-map --raname my-resource-adapter
--principalsmap eis-principal-1=server-principal-1,eis-principal-2
=server-principal-2,eis-principal-3=server-principal-1
workSecurityMap1
Command create-connector-work-security-map executed successfully.
```

Example 2 Creating a Connector Work Security Map (Group)

This example creates connector work security map `workSecurityMap2` that maps the backend EIS user group to the Payara Server user group.

```
asadmin create-connector-work-security-map --raname my-resource-adapter
--groupsmap eis-group-1=server-group-1,eis-group-2=server-group-2,
eis-group-3=server-group-1 workSecurityMap2
Command create-connector-work-security-map executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- delete-connector-work-security-map,
- list-connector-work-security-maps,
- update-connector-work-security-map

Create-Context-Service

The `create-context-service` asadmin command creates a context service resource.

Synopsis

```
asadmin [asadmin-options] create-context-service [--help]
        [--enabled={false|true}]
        [--contextinfoenabled={false|true}]
        [--contextinfo={Classloader|JNDI|Security|WorkArea}]]
        [--description description]
        [--property property]
        [--target target]
        jndi_name
```

Description

The `create-context-service` subcommand creates a context service resource.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--enabled`**: Determines whether the resource is enabled at runtime.

+ The default value is `true`. **`--contextinfoenabled`**: Determines whether container contexts are propagated to threads. If set to `true`, the contexts specified in the `--contextinfo` option are propagated. If set to `false`, no contexts are propagated and the `--contextinfo` option is ignored.

+ The default value is `true`. **`--contextinfo`**: Specifies individual container contexts to propagate to threads. Valid values are Classloader, JNDI, Security, and WorkArea. Values are specified in a comma-separated list and are case-insensitive.

+ All contexts are propagated by default. **`--description`**: Descriptive details about the resource. **`--property`**: Optional attribute name/value pairs for configuring the resource.

+ Payara Server does not define any additional properties for this resource. Moreover, this resource does not currently use any additional properties. **`--target`**: Specifies the target for which you are creating the resource. Valid targets are:

+ `server`;; Creates the resource for the default server instance.

+ This is the default value. `domain`;; Creates the resource for the domain. cluster_name;; Creates the resource for every server instance in the specified cluster. instance_name;; Creates the resource for the specified server instance.

Operands

jndi_name: The JNDI name of this resource.

Examples

Example 1 Creating a Context Service Resource

This example creates a context service resource named

`concurrent/myContextService`.

```
asadmin> create-context-service --description "context example" --property name
=Jose --target domain concurrent/myContextService
Context service concurrent/myContextService created successfully.
Command create-context-service executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- delete-context-service,
- list-context-services

Create-Custom-Resource

Creates a custom resource.

Synopsis

```
asadmin [asadmin-options] create-custom-resource [--help]
--restype type --factoryclass classname
[--enabled={true|false}] [--description text]
[--property (name=value)[:name=value]*] jndi-name
[--target target]
```

Description

The `create-custom-resource` subcommand creates a custom resource. A custom resource specifies a custom server-wide resource object factory that implements the `javax.naming.spi.ObjectFactory` interface.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--target`: The target on which the custom resource you are creating

will be available. Valid values are:
`server`;; The resource will be available on the default server instance and all domains hosted on the instance. This is the default value. `domain`;; The resource will be available on the specified domain only. cluster_name;; The resource will be available on every server instance in the cluster. instance_name;; The resource will be available on the specified server instance only.

NOTE

The resource is always created for the domain as a whole, but the `resource-ref` for the resource is only created for the specified `--target`. This means that although the resource is defined at the domain level, it is only available at the specified target level. Use the `create-resource-ref` subcommand to refer to the resource in multiple targets if needed.

`--restype`: The type of custom resource to be created. Specify a fully qualified type definition, for example `javax.naming.spi.ObjectFactory`. The resource type definition follows the format, xxx`.`xxx.--factoryclass`: Factory class name for the custom resource. This class implements the `javax.naming.spi.ObjectFactory` interface. **`--enabled`**: Determines whether the custom resource is enable at runtime. Default is true. **`--description`**: Text providing details about the custom resource. This description is a string value and can include a maximum of 250 characters. **`--property`**: Optional attribute name/value pairs for configuring the resource.

Operands

jndi-name: The JNDI name of this resource.

Examples

Example 1 Creating a Custom Resource

This example creates a custom resource.

```
asadmin> create-custom-resource --restype topic
--factoryclass com.imq.topic
Custom Resource mycustomresource created.
```

```
Command create-custom-resource executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- delete-custom-resource,
- list-custom-resources,
- create-resource-ref

Create-Deployment-Group

Creates a Payara Server deployment group.

Synopsis

```
asadmin [asadmin-options] create-deployment-group [--help]
name
```

Description

The `create-deployment-group` subcommand creates a Payara Server deployment group. Initially the deployment group contains no Payara Server instances, applications, or resources.

To add instances to the cluster, use the "`add-instance-to-deployment-group`".

- Deploying applications by using the deploy subcommand
- Creating resources by using subcommands such as create-jdbc-resource
- Creating references to applications that are already deployed in other targets by using the create-application-ref subcommand
- Creating references to resources that are already created in other targets by using the create-resource-ref subcommand

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: `-?`: Displays the help text for the subcommand.

Operands

name: The name of the deployment group.

+ The name must meet the following requirements:

+ * The name may contain only ASCII characters. * The name must start with a letter, a number, or an underscore. * The name may contain only the following characters: **Lowercase letters** Uppercase letters **Numbers** Hyphen **Period** Underscore * The name must be unique in the domain and must not be the name of another cluster, a named configuration, a Payara Server instance, or a node. * The name must not be `domain`, `server`, or any other keyword that is reserved by Payara Server.

Examples

Example 1 Creating a Deployment Group

This example creates a deployment group that is named `test-dgroup`:

```
asadmin create-deployment-group test-dgroup
Command create-deployment-group executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-application-ref,
- create-instance,
- create-jdbc-resource,

- create-local-instance,
- create-resource-ref,
- delete-deployment-group,
- delete-instance,
- delete-local-instance,
- deploy,
- list-deployment-groups,
- start-deployment-group,
- stop-deployment-group,
- restart-deployment-group,
- add-instance-to-deployment-group
- remove-instance-from-deployment-group

Create-Domain

Creates a domain.

Synopsis

```
asadmin [asadmin-options] create-domain [--help]
[--adminport adminport]
[--instanceport instanceport]
[--portbase portbase]
[--template template-name]
[--domaindir domaindir]
[--savemasterpassword={false|true}]
[--masterpasswordlocation filepath]
[--usemasterpassword={false|true}]
[--domainproperties (name=value)[:name=value]*]
[--keytooloptions (name=value)[:name=value]*]
[--savelogin={false|true}]
[--checkports={true|false}]
[--nopassword={false|true}]
[--hazelcastdasport hazelcastdasport]
[--hazelcaststartport hazelcaststartport]
[--hazelcastautoincrement hazelcastautoincrement]
domain-name
```

Description

The `create-domain` subcommand creates a Payara Server domain. A domain in Payara Server is an administrative namespace that complies with the Jakarta EE standard. Every domain has a configuration, which is stored in a set of files. Any number of domains, each of which has a distinct administrative identity, can be created in a given installation of Payara Server. A domain can exist independently of other domains.

Any user who has access to the `asadmin` utility on a given system can create a domain and store its configuration in a folder of the user's choosing. By default, the domain configuration is created in the default directory for domains. You can override this location to store the configuration elsewhere.

If domain customizers are found in JAR files in the `as-install/modules` directory when the `create-domain` subcommand is run, the customizers are processed. A domain customizer is a class that implements the `DomainInitializer` interface.

The `create-domain` subcommand creates a domain with a single administrative user specified by the `asadmin` utility option `--user`. If the `--user` option is not specified, and the `--nopassword` option is set to true, the default administrative user, `admin`, is used. If the `--nopassword` option is set to false (the default), a username is required. In this case, if you have not specified the user name by using the `--user` option, you are prompted to do so.

You choose an appropriate profile for the domain, depending on the applications that you want to run on your new domain. You can choose the developer, cluster, or enterprise profile for the domain you create. This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--adminport`**: The HTTP port or the HTTPS port for administration. This port is the port in the URL that you specify in your web browser to manage the domain, for example, `http://localhost:4949`. The `--adminport` option cannot be

used with the `--portbase` option. The default value is 4848.
The `--adminport` option overrides the `domain.adminPort` property of the `--domainproperties` option. **`--instanceport`**: The domain provides services so that applications can run when deployed. This HTTP port specifies where the web application context roots are available for a web browser to connect to. This port is a positive integer and must be available at the time of domain creation. The `--instanceport` option cannot be used with the `--portbase` option. The default value is 8080.
The `--instanceport` option overrides the `domain.instancePort` property of the `--domainproperties` option. **`--portbase`**: Determines the number with which port assignments should start. A domain uses a certain number of ports that are statically assigned. The portbase value determines where the assignment should start. The values for the ports are calculated as follows:
* Administration port: portbase + 48 * HTTP listener port: portbase + 80 * HTTPS listener port: portbase + 81 * JMS port: portbase + 76 * IIOP listener port: portbase + 37 * Secure IIOP listener port: portbase + 38 * Secure IIOP with mutual authentication port: portbase + 39 * JMX port: portbase + 86 * JPDA debugger port: portbase + 9 * Felix shell service port for OSGi module management: portbase + 66
When the `--portbase` option is specified, the output of this subcommand includes a complete list of used ports.
The `--portbase` option cannot be used with the `--adminport`, `--instanceport`, or the `--domainproperties` option. **`--template`**: The file name, including a relative or absolute path, of a domain configuration template to use for creating the domain. If a relative path is specified, the subcommand appends the path to the as-install`/lib/templates` directory to locate the file. If it is an absolute pathname, the subcommand locates the file in the specified path.
This option enables domains of different types to be created and custom domain templates to be defined. **`--domaindir`**: The directory where the domain is to be created. If specified, the path must be accessible in the filesystem. If not specified, the domain is created in the default domain directory, as-install`/domains`.
`--savemasterpassword`: Setting this option to `true` allows the master password to

be written to the file system. If this option is `true`, the `--usemasterpassword` option is also true, regardless of the value that is specified on the command line. The default value is `false`.

A master password is really a password for the secure key store. A domain is designed to keep its own certificate (created at the time of domain creation) in a safe place in the configuration location. This certificate is called the domain's SSL server certificate. When the domain is contacted by a web browser over a secure channel (HTTPS), this certificate is presented by the domain. The master password is supposed to protect the store (a file) that contains this certificate. This file is called `keystore.jks` and is created in the configuration directory of the domain created. If however, this option is chosen, the master password is saved on the disk in the domain's configuration location. The master password is stored in a file called `master-password`, which is a Java JCEKS type keystore. The reason for using the `--savemasterpassword` option is for unattended system boots. In this case, the master password is not prompted for when the domain starts because the password will be extracted from this file.

It is best to create a master password when creating a domain, because the master password is used by the `start-domain` subcommand. For security purposes, the default setting should be false, because saving the master password on the disk is an insecure practice, unless file system permissions are properly set. If the master password is saved, then `start-domain` does not prompt for it. The master password gives an extra level of security to the environment. **`--masterpasswordlocation`**: Only applies if the `--savemasterpassword` parameter is set to true. This parameter receives an absolute filepath that the master password will be saved to.

If this parameter is omitted, the default location will be used. **`--usemasterpassword`**: Specifies whether the key store is encrypted with a master password that is built into the system or a user-defined master password.

If `false` (default), the keystore is encrypted with a well-known password that is built into the system. Encrypting the keystore with a password that is built into the system provides no additional security.

If `true`, the subcommand obtains the master password from the `AS_ADMIN_MASTERPASSWORD` entry in the password file or prompts for the master

password. The password file is specified in the `--passwordfile` option of the asadmin-1mutility.

If the `--savemasterpassword` option is `true`, this option is also true, regardless of the value that is specified on the command line. **`--domainproperties`**: Setting the optional name/value pairs overrides the default values for the properties of the domain to be created. The list must be separated by the colon (:) character. The `--portbase` options cannot be used with the `--domainproperties` option. The following properties are available:

`domain.adminPort`;; This property specifies the port number of the HTTP port or the HTTPS port for administration. This port is the port in the URL that you specify in your web browser to manage the instance, for example, `http://localhost:4949`. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges.

The `domain.adminPort` property is overridden by the `--adminport` option.

`domain.instancePort`;; This property specifies the port number of the port that is used to listen for HTTP requests. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges.

The `domain.instancePort` property is overridden by `--instanceport` option.

`domain.jmxPort`;; This property specifies the port number on which the JMX connector listens. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `http.ssl.port`;; This property specifies the port number of the port that is used to listen for HTTPS requests. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `java.debugger.port`;; This property specifies the port number of the port that is used for connections to the Java Platform Debugger Architecture (JPDA) debugger. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `jms.port`;; This property specifies the port number for the Java Message Service provider. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges.

`orb.listener.port`;; This property specifies the port number of the port that is used for IIOP connections. Valid values are 1-65535. On UNIX, creating sockets that listen on

ports 1-1024 requires superuser privileges. `orb.mutualauth.port`;; This property specifies the port number of the port that is used for secure IIOP connections with client authentication. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `orb.ssl.port`;; This property specifies the port number of the port that is used for secure IIOP connections. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `osgi.shell.telnet.port`;; This property specifies the port number of the port that is used for connections to the Apache Felix Remote Shell. This shell uses the Felix shell service to interact with the OSGi module management subsystem. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `hazelcast.das.port`;; This property specifies the port number of the port the DAS use to run the Data Grid (Hazelcast) on. The `hazelcast.das.port` property is overridden by the `--hazelcastdasport` option. `hazelcast.start.port`;; This property specifies the port number of the port the other Payara Server instances use to run the Data Grid (Hazelcast) on. The `hazelcast.start.port` property is overridden by the `--hazelcaststartport` option. `hazelcast.auto.increment`;; This property specifies whether to use the next unoccupied port that is available starting with the start port. When auto-increment is turned off an occupied start port results in a startup failure instead. The `hazelcast.auto.increment` property is overridden by the `--hazelcastautoincrement` option. **`--keytooloptions`**: Specifies an optional list of name-value pairs of keytool options for a self-signed server certificate. The certificate is generated during the creation of the domain. Each pair in the list must be separated by the colon (:) character.

+ Allowed options are as follows:

+ `CN`;; Specifies the common name of the host that is to be used for the self-signed certificate. This option name is case insensitive.

+ By default, the name is the fully-qualified name of the host where the `create-domain` subcommand is run. **`--savelogin`**: If set to true, this option saves the

administration user name and password. Default value is false. The username and password are stored in the `.asadminpass` file in user's home directory. A domain can only be created locally. Therefore, when using the `--savelogin` option, the host name saved in `.asadminpass` is always `localhost`. If the user has specified default administration port while creating the domain, there is no need to specify `--user`, `--passwordfile`, `--host`, or `--port` on any of the subsequent `asadmin` remote commands. These values will be obtained automatically.

NOTE	When the same user creates multiple domains that have the same administration port number on the same or different host (where the home directory is NFS mounted), the subcommand does not ask if the password should be overwritten. The password will always be overwritten.

`--checkports`: Specifies whether to check for the availability of the administration, HTTP, JMS, JMX, and IIOP ports. The default value is `true`. **`--nopassword`**: Specifies whether the administrative user will have a password. If false (the default), the password is specified by the `AS_ADMIN_PASSWORD` entry in the `asadmin` password file (set by using the `--passwordfile` option). If false and the `AS_ADMIN_PASSWORD` is not set, you are prompted for the password.

+ If `true`, the administrative user is created without a password.

+ WARNING: If a username for the domain is not specified by using the `--user` option, and the `--nopassword` option is set to `true`, the default username, `admin`, is used.

`--hazelcastdasport`: The port to run the Data Grid (Hazelcast) on for the DAS. If this port is busy and auto-increment option is enabled, the port specified will be incremented until a valid port is found.

+ The `--hazelcastdasport` option cannot be used with the `--portbase` option.

+ The `--hazelcastdasport` option overrides the `hazelcast.das.port` property of the `--domainproperties` option. The default value is `4900`.

`--hazelcaststartport`: The port the other Payara Server instances use to run the Data Grid (Hazelcast) on.

+ If this port is busy and the `--hazelcastautoincrement` option is enabled, the port specified will be incremented until a valid port is found. The `--hazelcaststartport` option cannot be used with the `--portbase` option. The `--hazelcaststartport` option overrides the `hazelcast.start.port` property of the `--domainproperties` option.

+ The default value is `5900`.

+ NOTE: If the port base options are used, the values for the ports of the Hazelcast DAS start port and the Hazelcast start Port are calculated as follows:
Hazelcast DAS start port = `portbase + 49` and Hazelcast Instance start port= `portbase + 59`.

`--hazelcastautoincrement`: By default, the Data Grid (Hazelcast) uses the next unoccupied port that is available starting with the start port.

+ When auto-increment is turned off an occupied start port results in a startup failure instead. The `--hazelcastautoincrement` option overrides the `hazelcast.auto.inrement` property of the `--domainproperties` option.

+ The default value is `5900`.

Operands

domain-name: The name of the domain to be created. The name may contain only ASCII characters and must be a valid directory name for the operating system on the host where the domain is created.

Examples

Example 1 Creating a Domain

This example creates a domain named `domain4`.

```
asadmin> create-domain --adminport 4848 domain4
```

```
Enter admin user name [Enter to accept default "admin" / no password]>
Using port 4848 for Admin.
Using default port 8080 for HTTP Instance.
Using default port 7676 for JMS.
Using default port 3700 for IIOP.
Using default port 8181 for HTTP_SSL.
Using default port 3820 for IIOP_SSL.
Using default port 3920 for IIOP_MUTUALAUTH.
Using default port 8686 for JMX_ADMIN.
Using default port 6666 for OSGI_SHELL.
Distinguished Name of the self-signed X.509 Server Certificate is:
[CN=sr1-usca-22,OU=Payara,O=Payara Foundation,L==Great Malvern,ST=Worcestershire,C
=UK]
No domain initializers found, bypassing customization step
Domain domain4 created.
Domain domain4 admin port is 4848.
Domain domain4 allows admin login as user "admin" with no password.
Command create-domain executed successfully.
```

Example 2 Creating a Domain in an Alternate Directory

This example creates a domain named `sampleDomain` in the `/home/someuser/domains` directory.

```
asadmin> create-domain --domaindir /home/someuser/domains --adminport 7070
--instanceport 7071 sampleDomain
Enter admin user name [Enter to accept default "admin" / no password]>
Using port 7070 for Admin.
Using port 7071 for HTTP Instance.
Using default port 7676 for JMS.
Using default port 3700 for IIOP.
Using default port 8181 for HTTP_SSL.
Using default port 3820 for IIOP_SSL.
Using default port 3920 for IIOP_MUTUALAUTH.
Using default port 8686 for JMX_ADMIN.
Using default port 6666 for OSGI_SHELL.
Enterprise ServiceDistinguished Name of the self-signed X.509 Server Certificate
is:
[CN=sr1-usca-22,OU=Payara,O=Payara Foundation,L==Great Malvern,ST=Worcestershire,C
=UK]
No domain initializers found, bypassing customization step
Domain sampleDomain created.
Domain sampleDomain admin port is 7070.
Domain sampleDomain allows admin login as user "admin" with no password.
Command create-domain executed successfully.
```

Example 3 Creating a Domain and Saving the Administration User Name and Password

This example creates a domain named `myDomain` and saves the administration username and password.

```
asadmin> create-domain --adminport 8282 --savelogin=true myDomain
Enter the admin password [Enter to accept default of no password]>
Enter the master password [Enter to accept default password "changeit"]>
Using port 8282 for Admin.
Using default port 8080 for HTTP Instance.
Using default port 7676 for JMS.
Using default port 3700 for IIOP.
Using default port 8181 for HTTP_SSL.
Using default port 3820 for IIOP_SSL.
Using default port 3920 for IIOP_MUTUALAUTH.
Using default port 8686 for JMX_ADMIN.
Using default port 6666 for OSGI_SHELL.
Enterprise ServiceDistinguished Name of the self-signed X.509 Server Certificate
is:
[CN=sr1-usca-22,OU=Payara,O=Payara Foundation,L==Great Malvern,ST=Worcestershire,C
=UK]
No domain initializers found, bypassing customization step
Domain myDomain created.
Domain myDomain admin port is 8282.
Domain myDomain allows admin login as user "admin" with no password.
Login information relevant to admin user name [admin]
for this domain [myDomain] stored at
[/home/someuser/.asadminpass] successfully.
Make sure that this file remains protected.
Information stored in this file will be used by
asadmin commands to manage this domain.
Command create-domain executed successfully.
```

Example 4 Creating a Domain and Designating the Certificate Host

This example creates a domain named `domain5`. The common name of the host that is to be used for the self-signed certificate is `trio`.

```
asadmin> create-domain --adminport 9898 --keytooloptions CN=trio domain5
Enter the admin password [Enter to accept default of no password]>
Enter the master password [Enter to accept default password "changeit"]>
Using port 9898 for Admin.
Using default port 8080 for HTTP Instance.
Using default port 7676 for JMS.
Using default port 3700 for IIOP.
Using default port 8181 for HTTP_SSL.
Using default port 3820 for IIOP_SSL.
Using default port 3920 for IIOP_MUTUALAUTH.
Using default port 8686 for JMX_ADMIN.
Using default port 6666 for OSGI_SHELL.
Distinguished Name of the self-signed X.509 Server Certificate is:
[CN=trio,OU=Payara,O=Payara Foundation.,L==Great Malvern,ST=Worcestershire,C=UK]
No domain initializers found, bypassing customization step
Domain domain5 created.
Domain domain5 admin port is 9898.
Domain domain5 allows admin login as user "admin" with no password.
Command create-domain executed successfully.
```

Example 5 Creating a Domain with Hazelcast specific domain properties

This example creates a domain named `testDomain` with the port to run Hazelcast on for the DAS set to 7900 and the port the other Payara Server instances use to run Hazelcast set to 8900.

```
asadmin> asadmin create-domain --hazelcastdasport 7900 --hazelcaststartport 8900
testDomain
Using default port 4848 for Admin.
Using default port 8080 for HTTP Instance.
Using default port 7676 for JMS.
Using default port 3700 for IIOP.
Using default port 8181 for HTTP_SSL.
Using default port 3820 for IIOP_SSL.
Using default port 3920 for IIOP_MUTUALAUTH.
Using default port 8686 for JMX_ADMIN.
Using default port 6666 for OSGI_SHELL.
Using default port 9009 for JAVA_DEBUGGER.
Using port 7900 for Hazelcast DAS.
Using port 8900 for Hazelcast Start.
Distinguished Name of the self-signed X.509 Server Certificate is:
[CN=dev,OU=Payara,O=Payara Foundation,L=Great Malvern,ST=Worcestershire,C=UK]
Distinguished Name of the self-signed X.509 Server Certificate is:
[CN=dev-instance,OU=Payara,O=Payara Foundation,L=Great Malvern,ST=Worcestershire,C
=UK]
Domain testDomain created.
Domain testDomain admin port is 4848.
Domain testDomain allows admin login as user "admin" with no password.
Command create-domain executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- delete-domain,
- list-domains,
- login,
- start-domain,
- stop-domain
- Apache Felix Remote Shell

(https://felix.apache.org/documentation/subprojects/apache-felix-remote-shell.html)

- Java Platform Debugger Architecture (JPDA) (https://docs.oracle.com/javase/8/docs/technotes/guides/jpda/)

Create-File-User

Creates a new file user.

Synopsis

```
asadmin [asadmin-options] create-file-user [--help]
[--authrealmnameauth_realm_name]
[--target target
[--groups user_groups[:user_groups]*] user_name
```

Description

The `create-file-user` subcommand creates an entry in the keyfile with the specified username, password, and groups. Multiple groups can be created by separating them with a colon (`:`). If auth_realm_name is not specified, an entry is created in the keyfile for the default realm.

If auth_realm_name is specified, an entry is created in the keyfile using the `auth_realm_name`.

You can use the `--passwordfile` option of the asadmin-1m command to specify the password for the user. The password file entry must be of the form `AS_ADMIN_USERPASSWORD`=user-password.

If a password is not provided, this subcommand fails if secure administration is enabled and the user being created is an administrative user.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This is the name of the target on which the command

operates. The valid targets are config, instance, cluster, or server. By default, the target is the server. `--groups`: This is the group associated with this file user. `--authrealmname`: The name of the realm in which the new user is created. If you do not specify this option, the user is created in the "file" realm.

Operands

user_name: This is the name of file user to be created.

Examples

Example 1 Creating a User in the File Realm

This example creates a file realm user named `sample_user`. It is assumed that an authentication realm has already been created using the `create-auth-realm` subcommand.

```
asadmin> create-file-user
--groups staff:manager
--authrealmname auth-realm1 sample_user
Command create-file-user executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-auth-realm,
- delete-file-user,
- list-file-groups,
- list-file-users,
- update-file-user

Create-Http-Listener

Adds a new HTTP network listener socket.

Synopsis

```
asadmin [asadmin-options] create-http-listener [--help]
--listeneraddress address
--listenerport listener-port
{--default-virtual-server} virtual-server
[--servername server-name]
[--acceptorthreads acceptor-threads]
[--securityenabled={false|true}]
[--enabled={true|false}]
[--target target]
listener-id
```

Description

The `create-http-listener` subcommand creates an HTTP network listener. This subcommand is supported in remote mode only.

NOTE

If you edit the special HTTP network listener named `admin-listener`, you must restart the server for the changes to take effect. The Administration Console does not tell you that a restart is required in this case.

NOTE

This subcommand is provided for backward compatibility and as a shortcut for creating network listeners that use the HTTP protocol. Behind the scenes, this subcommand creates a network listener and its associated protocol, transport, and HTTP configuration.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--listeneraddress`**: The IP address or the hostname (resolvable by DNS). **`--listenerport`**: The port number to create the listen socket on. Legal values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. Configuring an SSL listen socket to listen on port 443 is recommended. **`--default-virtual-server`**: The ID attribute of the default virtual server for this listener. **`--servername`**: Tells the server what to put in the host name section of any URLs it sends to the client. This affects URLs the server automatically generates; it

doesn't affect the URLs for directories and files stored in the server. This name should be the alias name if your server uses an alias. If a colon and port number are appended, that port will be used in URLs that the server sends to the client. `--acceptorthreads`: The number of acceptor threads for the listener socket. The recommended value is the number of processors in the machine. The default value is 1. `--securityenabled`: If set to true, the HTTP listener runs SSL. You can turn SSL2 or SSL3 ON or OFF and set ciphers using an SSL element. The security setting globally enables or disables SSL by making certificates available to the server instance. The default value is `false`. `--enabled`: If set to true, the listener is enabled at runtime. The default value is `true`. `--target`: Creates the HTTP listener only on the specified target. Valid values are as follows:
`server`;; Creates the HTTP listener on the default server instance. This is the default value. configuration-name;; Creates the HTTP listener in the specified configuration. cluster-name;; Creates the HTTP listener on all server instances in the specified cluster. standalone-instance-name;; Creates the HTTP listener on the specified standalone server instance.

Operands

listener-id: The listener ID of the HTTP network listener.

Examples

Example 1 Creating an HTTP Network Listener

The following command creates an HTTP network listener named `sampleListener` that uses a non-default number of acceptor threads and is not enabled at runtime:

```
asadmin> create-http-listener --listeneraddress 0.0.0.0 --listenerport 7272
--default-virtual-server server --servername host1.sun.com --acceptorthreads 100
--securityenabled=false --enabled=false sampleListener
Command create-http-listener executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-ssl,
- create-network-listener,
- set-network-listener-configuration,
- create-virtual-server,
- delete-http-listener,
- list-http-listeners,
- restart-http-listeners

Create-Http-Redirect

Adds a new HTTP redirect.

Synopsis

```
asadmin [asadmin-options] create-http-redirect [--help]
[--redirect-port redirect-port]
[--secure-redirect={false|true}]
[--target target]
protocol-name
```

Description

The `create-http-redirect` subcommand creates an HTTP redirect. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--redirect-port`**: Port number for redirects. If the HTTP listener is supporting non-SSL requests, and a request is received for which a matching security-constraint requires SSL transport, Payara Server automatically redirects the request to this port number. **`--secure-redirect`**: If set to true, the HTTP redirect runs SSL. The default value is `false`. **`--target`**: Creates the HTTP redirect only on the specified target. Valid values are as follows:
`server`;; Creates the HTTP redirect on the default server instance. This is the default

value. configuration-name;; Creates the HTTP redirect in the specified configuration. cluster-name;; Creates the HTTP redirect on all server instances in the specified cluster. standalone-instance-name;; Creates the HTTP redirect on the specified standalone server instance.

Operands

protocol-name: The name of the protocol to which to apply the redirect.

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- delete-http-redirect

Create-Http

Sets HTTP parameters for a protocol.

Synopsis

```
asadmin [asadmin-options] create-http [--help]
--default-virtual-server virtual-server
[--request-timeout-seconds timeout]
[--timeout-seconds timeout]
[--max-connection max-keepalive]
[--dns-lookup-enabled={false|true}]
[--servername server-name]
[--target target]
protocol-name
```

Description

The `create-http` subcommand creates a set of HTTP parameters for a protocol, which in turn configures one or more network listeners. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the

subcommand. `--default-virtual-server`: The ID attribute of the default virtual server for the associated network listeners. `--request-timeout-seconds`: The time in seconds at which the request times out. If you do not set this option, the request times out in 30 seconds. `--timeout-seconds`: The maximum time in seconds for which a keep alive connection is kept open. A value of `0` or less means keep alive connections are kept open indefinitely. The default is `30`. `--max-connection`: The maximum number of HTTP requests that can be pipelined until the connection is closed by the server. Set this property to `1` to disable HTTP/1.0 keep-alive, as well as HTTP/1.1 keep-alive and pipelining. The default is `256`. `--dns-lookup-enabled`: If set to `true`, looks up the DNS entry for the client. The default is `false`. `--servername`: Tells the server what to put in the host name section of any URLs it sends to the client. This affects URLs the server automatically generates; it doesn't affect the URLs for directories and files stored in the server. This name should be the alias name if your server uses an alias. If a colon and port number are appended, that port will be used in URLs that the server sends to the client. `--target`: Creates the set of HTTP parameters only on the specified target. Valid values are as follows: `server`;; Creates the set of HTTP parameters on the default server instance. This is the default value. configuration-name;; Creates the set of HTTP parameters in the specified configuration. cluster-name;; Creates the set of HTTP parameters on all server instances in the specified cluster. standalone-instance-name;; Creates the set of HTTP parameters on the specified standalone server instance.

Operands

protocol-name: The name of the protocol to which this HTTP parameter set applies.

Examples

Example 1 Using the `create-http` Subcommand

The following command creates an HTTP parameter set for the protocol named `http-1`:

```
asadmin> create-http --timeout-seconds 60 --default-virtual-server server http-1
Command create-http executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-network-listener,
- create-protocol,
- create-virtual-server,
- delete-http

Create-Iiop-Listener

Adds an IIOP listener.

Synopsis

```
asadmin [asadmin-options] create-iiop-listener [--help]
--listeneraddress address
[--iiopport iiop-port-number] [--securityenabled={false|true}] [--enabled={
true|false}]
[--property (name=value)[:name=value]*]
[--target target] listener_id
```

Description

The `create-iiop-listener` subcommand creates an IIOP listener. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--listeneraddress`: Either the IP address or the hostname (resolvable by DNS).

`--iiopport`: The IIOP port number. The default value is 1072.

`--securityenabled`: If set to true, the IIOP listener runs SSL. You can turn SSL2 or

SSL3 ON or OFF and set ciphers using an SSL element. The security setting globally enables or disables SSL by making certificates available to the server instance. The default value is `false`.

`--enabled`: If set to true, the IIOP listener is enabled at runtime. The default value is `true`.

`--property`: Optional attribute name/value pairs for configuring the IIOP listener.

`--lazyInit`: Whether to lazily initialise the listener. The default value is `false`.

`--target`: Specifies the target for which you are creating the IIOP listener. Valid values are:
`server`;; Creates the listener for the default server instance `server` and is the default value. configuration_name;; Creates the listener for the named configuration. cluster_name;; Creates the listener for every server instance in the cluster. stand-alone_instance_name;; Creates the listener for a particular standalone server instance.

Operands

listener_id: A unique identifier for the IIOP listener to be created.

Examples

Example 1 Creating an IIOP Listener

The following command creates an IIOP listener named `sample_iiop_listener`:

```
asadmin> create-iiop-listener --listeneraddress 192.168.1.100
--iiopport 1400 sample_iiop_listener
Command create-iiop-listener executed successfully.
```

Example 2 Creating an IIOP Listener with a Target Cluster

The following command creates an IIOP listener named `iiop_listener_2` for the cluster `mycluster`. It uses the target option.

```
asadmin> create-iiop-listener --listeneraddress 0.0.0.0 --iiopport 1401
--target mycluster iiop_listener_2
```

```
Command create-iiop-listener executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-ssl,
- delete-iiop-listener,
- list-iiop-listeners

Create-Instance

Creates a Payara instance.

Synopsis

```
asadmin [asadmin-options] create-instance [--help]
--node node-name
[--config config-name | --cluster cluster-name]
[--lbenabled={true|false}]
[--portbase=port-number] [--checkports={true|false}]
[--systemproperties (name=value)[:name=value]* ]
[--dataGridStartPort startPort]
instance-name
```

Description

The `create-instance` subcommand creates a Payara instance. This subcommand requires the secure shell (SSH) to be configured on the host where the domain administration server (DAS) is running and on the host that is represented by the node where the instance is to reside.

NOTE	SSH is not required if the instance is to reside on a node of type `CONFIG` that represents the local host. A node of type `CONFIG` is not enabled for remote communication over SSH.

You may run this command from any host that can contact the DAS.

A Payara Server instance is a single Virtual Machine for the Java platform (Java Virtual Machine or JVM machine) on a single node in which Payara is running. A node defines the host where the Payara Server instance resides. The JVM machine must be compatible with the Java Platform, Enterprise Edition (Java EE).

A Payara Server instance requires a reference to the following items:

- The node that defines the host where the instance resides. The node must be specified in the command to create the instance.
- The named configuration that defines the configuration of the instance. The configuration can be specified in the command to create the instance, but is not required. If no configuration is specified for an instance that is not joining a cluster, the subcommand creates a configuration for the instance. An instance that is joining a cluster receives its configuration from its parent cluster.

Each Payara Server instance is one of the following types of instance:

Standalone instance: A standalone instance does not share its configuration with any other instances or clusters. A standalone instance is created if either of the following conditions is met:

+ * No configuration or cluster is specified in the command to create the instance. * A configuration that is not referenced by any other instances or clusters is specified in the command to create the instance.

+ When no configuration or cluster is specified, a copy of the `default-config` configuration is created for the instance. The name of this configuration is instance-name`-config`, where instance-name represents the name of an un-clustered server instance.

Shared instance: A shared instance shares its configuration with other instances or clusters. A shared instance is created if a configuration that is referenced by other instances or clusters is specified in the command to create the instance. **Clustered instance**: A clustered instance inherits its configuration from the cluster to which the

instance belongs and shares its configuration with other instances in the cluster. A clustered instance is created if a cluster is specified in the command to create the instance.

+ Any instance that is not part of a cluster is considered an un-clustered server instance. Therefore, standalone instances and shared instances are un-clustered server instances.

By default, this subcommand attempts to resolve possible port conflicts for the instance that is being created. The subcommand also assigns ports that are currently not in use and not already assigned to other instances on the same node. The subcommand assigns these ports on the basis of an algorithm that is internal to the subcommand.

Use the `--systemproperties` option to resolve port conflicts for additional instances on the same node. System properties of an instance can be manipulated by using the create-system-properties subcommand and the delete-system-property subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--node`**: The name of the node that defines the host where the instance is to be created. The node must already exist. If the instance is to be created on the host where the domain administration server (DAS) is running, use the predefined node `localhost-`domain.--config`: Specifies the named configuration that the instance references. The configuration must exist and must not be named `default-config` or `server-config`. Specifying the `--config` option creates a shared instance. The `--config` option and the `--cluster` option are mutually exclusive. If both options are omitted, a standalone instance is created. **`--cluster`**: Specifies the cluster from which the instance inherits its configuration. Specifying the `--cluster` option creates a clustered instance. The `--config` option and the `--cluster` option

are mutually exclusive. If both options are omitted, a standalone instance is created. **`--lbenabled`**: Specifies whether the instance is enabled for load balancing. Possible values are as follows:
`true`;; The instance is enabled for load balancing (default). When an instance is enabled for load balancing, a load balancer sends requests to the instance. `false`;; The instance is disabled for load balancing. When an instance is disabled for load balancing, a load balancer does not send requests to the instance. **`--portbase`**: Determines the number with which the port assignment should start. An instance uses a certain number of ports that are statically assigned. The `portbase` value determines where the assignment should start. The values for the ports are calculated as follows:

+ * Administration port: `portbase + 48` * HTTP listener port: `portbase + 80` * HTTPS listener port: `portbase + 81` * JMS port: `portbase + 76` * IIOP listener port: `portbase + 37` * Secure IIOP listener port: `portbase + 38` * Secure IIOP with mutual authentication port: `portbase + 39` * JMX port: `portbase + 86` * JPA debugger port: `portbase + 9` * Felix shell service port for OSGi module management: `portbase + 66`

+ When the `--portbase` option is specified, the output of this subcommand includes a complete list of used ports. **`--checkports`**: Specifies whether to check for the availability of the administration, HTTP, JMS, JMX, and IIOP ports. The default value is `true`. **`--systemproperties`**: Defines system properties for the instance. These properties override property definitions for port settings in the instance's configuration. Predefined port settings must be overridden if, for example, two clustered instances reside on the same host. In this situation, port settings for one instance must be overridden because both instances share the same configuration. The following properties are available:
`ASADMIN_LISTENER_PORT`;; This property specifies the port number of the HTTP port or HTTPS port through which the DAS connects to the instance to manage the instance. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `HTTP_LISTENER_PORT`;; This property specifies the port number of the port that is used to listen for HTTP requests. Valid values are 1-65535. On UNIX,

creating sockets that listen on ports 1-1024 requires superuser privileges. `HTTP_SSL_LISTENER_PORT`;; This property specifies the port number of the port that is used to listen for HTTPS requests. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `IIOP_LISTENER_PORT`;; This property specifies the port number of the port that is used for IIOP connections. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `IIOP_SSL_LISTENER_PORT`;; This property specifies the port number of the port that is used for secure IIOP connections. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `IIOP_SSL_MUTUALAUTH_PORT`;; This property specifies the port number of the port that is used for secure IIOP connections with client authentication. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `JAVA_DEBUGGER_PORT`;; This property specifies the port number of the port that is used for connections to the Java Platform Debugger Architecture (JPDA)(http://www.oracle.com/technetwork/java/javase/tech/jpda-141715.html) debugger. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `JMS_PROVIDER_PORT`;; This property specifies the port number for the Java Message Service provider. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `JMX_SYSTEM_CONNECTOR_PORT`;; This property specifies the port number on which the JMX connector listens. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `OSGI_SHELL_TELNET_PORT`;; This property specifies the port number of the port that is used for connections to the Apache Felix Remote Shell (`http://felix.apache.org/site/apache-felix-remote-shell.html`). This shell uses the Felix shell service to interact with the OSGi module management subsystem. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. **`--dataGridStartPort`**: Sets Data Grid (Hazelcast) instance's starting port.

+ If set to `0` the Domain wide start port will be used instead.

+ The default value is `0`.

Operands

instance-name: The name of the instance that is being created.

+ The name must meet the following requirements:

+ * The name may contain only ASCII characters. * The name must start with a letter, a number, or an underscore. * The name may contain only the following characters: **Lowercase letters** Uppercase letters **Numbers** Hyphen **Period** Underscore * The name must be unique in the domain and must not be the name of another Payara instance, a cluster, a named configuration, or a node. * The name must not be `domain`, `server`, or any other keyword that is reserved by Payara Server.

Examples

Example 1 Creating a Standalone Payara Instance

This example creates the standalone Payara instance `pmdsainst` in the domain `domain1` on the local host.

```
asadmin> create-instance --node localhost-domain1 pmdsainst
Port Assignments for server instance pmdsainst:
JMX_SYSTEM_CONNECTOR_PORT=28688
JMS_PROVIDER_PORT=27678
ASADMIN_LISTENER_PORT=24850
HTTP_LISTENER_PORT=28082
IIOP_LISTENER_PORT=23702
IIOP_SSL_LISTENER_PORT=23822
HTTP_SSL_LISTENER_PORT=28183
IIOP_SSL_MUTUALAUTH_PORT=23922

Command create-instance executed successfully.
```

Example 2 Creating a Standalone Payara Instance With Custom Port Assignments

This example creates the standalone Payara instance `pmdcpinst` in the domain `domain1` on the local host. Custom port numbers are assigned to the following ports:

- HTTP listener port
- HTTPS listener port

- IIOP connections port
- Secure IIOP connections port
- Secure IIOP connections port with mutual authentication
- JMX connector port

```
asadmin> create-instance --node localhost-domain1
--systemproperties HTTP_LISTENER_PORT=58294:
HTTP_SSL_LISTENER_PORT=58297:
IIOP_LISTENER_PORT=58300:
IIOP_SSL_LISTENER_PORT=58303:
IIOP_SSL_MUTUALAUTH_PORT=58306:
JMX_SYSTEM_CONNECTOR_PORT=58309 pmdcpinst
Port Assignments for server instance pmdcpinst:
JMS_PROVIDER_PORT=27679
ASADMIN_LISTENER_PORT=24851

Command create-instance executed successfully.
```

Example 3 Creating a Shared Payara Server Instance

This example creates the shared Payara instance `pmdsharedinst1` in the domain `domain1` on the local host. The shared configuration of this instance is `pmdsharedconfig`.

```
asadmin> create-instance --node localhost-domain1 --config pmdsharedconfig
pmdsharedinst1
Port Assignments for server instance pmdsharedinst1:
JMX_SYSTEM_CONNECTOR_PORT=28687
JMS_PROVIDER_PORT=27677
ASADMIN_LISTENER_PORT=24849
HTTP_LISTENER_PORT=28081
IIOP_LISTENER_PORT=23701
IIOP_SSL_LISTENER_PORT=23821
HTTP_SSL_LISTENER_PORT=28182
IIOP_SSL_MUTUALAUTH_PORT=23921

Command create-instance executed successfully.
```

Example 4 Creating a Clustered Payara Server Instance

This example creates the clustered Payara instance `pmdinst1` in the domain `domain1` on the local host. The instance is a member of the cluster `pmdclust1`.

```
asadmin> create-instance --node localhost-domain1 --cluster pmdclust pmdinst1
```

```
Port Assignments for server instance pmdinst1:
JMX_SYSTEM_CONNECTOR_PORT=28686
JMS_PROVIDER_PORT=27676
HTTP_LISTENER_PORT=28080
ASADMIN_LISTENER_PORT=24848
IIOP_SSL_LISTENER_PORT=23820
IIOP_LISTENER_PORT=23700
HTTP_SSL_LISTENER_PORT=28181
IIOP_SSL_MUTUALAUTH_PORT=23920

Command create-instance executed successfully.
```

Example 5 Creating a Standalone Payara Server Instance with Specific Data Grid Start Port

This example creates a Standalone Payara instance `instance1` in the domain `domain1` on the local host. Setting the Data Grid Start Port when an instance is created to `2900`.

```
asadmin> create-instance --dataGridStartPort 2900 --node localhost-domain1
instance1
Command _create-instance-filesystem executed successfully.
Port Assignments for server instance instance1:
OSGI_SHELL_TELNET_PORT=26666
JAVA_DEBUGGER_PORT=29009
JMS_PROVIDER_PORT=27676
HTTP_LISTENER_PORT=28080
IIOP_SSL_LISTENER_PORT=23820
ASADMIN_LISTENER_PORT=24848
IIOP_SSL_MUTUALAUTH_PORT=23920
JMX_SYSTEM_CONNECTOR_PORT=28686
HTTP_SSL_LISTENER_PORT=28181
IIOP_LISTENER_PORT=23700
The instance, instance1, was created on host localhost
Command create-instance executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-local-instance,
- create-node-config,

- create-node-ssh,
- create-system-properties,
- delete-instance,
- delete-system-property,
- list-instances,
- setup-ssh,
- start-instance,
- stop-instance

Create-Jacc-Provider

Enables administrators to create a JACC provider that can be used by third-party authorization modules for applications running in Payara Server.

Synopsis

```
asadmin [asadmin-options] create-jacc-provider [--help]
[--policyproviderclass pol-provider-class]
[--policyconfigfactoryclass pc-factory-class]
[--property name=value)[:name=value]*]
[--target target] jacc-provider-name
```

Description

The `create-jacc-provider` subcommand creates a JSR-115—compliant Java Authorization Contract for Containers (JACC) provider that can be used for authorization of applications running in Payara. The JACC provider is created as a `jacc-provider` element within the `security-service` element in the domain's `domain.xml` file.

The default Payara installation includes two JACC providers, named `default` and `simple`. Any JACC providers created with the `create-jacc-provider` subcommand are in addition to these two default providers. The default Payara JACC providers implement a simple, file-based authorization engine that complies with the JACC specification. The `create-jacc-provider` subcommand makes it possible to specify additional third-party JACC providers.

You can create any number of JACC providers within the `security-service` element, but the Payara runtime uses only one of them at any given time. The `jacc-provider` element in the `security-service` element points to the name of the provider that is currently in use by Payara. If you change this element to point to a different JACC provider, restart Payara.

This command is supported in remote mode only.

Options

If an option has a short option name, then the short option precedes the long option name. Short options have one dash whereas long options have two dashes.

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--policyproviderclass`**: Specifies the fully qualified class name for the `javax.security.jacc.policy.provider` that implements the `java.security.Policy`. **`--policyconfigfactoryclass`**: Specifies the fully qualified class name for the `javax.security.jacc.PolicyConfigurationFactory.provider` that implements the provider-specific `javax.security.jacc.PolicyConfigurationFactory`. **`--property`**: Optional attribute name/value pairs for configuring the JACC provider. The following properties are available:
`repository`;; The directory containing the JACC policy file. For the `default` Payara JACC provider, the default directory is `${com.sun.aas.instanceRoot}/generated/policy`. This property is not defined by default for the `simple` Payara JACC provider. **`--target`**: Specifies the target for which you are creating the JACC provider. The following values are valid:
`server`;; Creates the JACC provider on the default server instance. This is the default value. configuration_name;; Creates the JACC provider in the specified configuration. cluster_name;; Creates the JACC provider on all server instances in the specified cluster. instance_name;; Creates the JACC provider on a specified server instance.

Operands

jacc-provider-name: The name of the provider used to reference the `jacc-provider` element in `domain.xml`.

Examples

Example 1 Creating a JACC Provider

The following example shows how to create a JACC provider named `testJACC` on the default `server` target.

```
asadmin> create-jacc-provider
--policyproviderclass org.glassfish.exousia.modules.locked.SimplePolicyProvider
--policyconfigfactoryclass
org.glassfish.exousia.modules.locked.SimplePolicyConfigurationFactory
testJACC

Command create-jacc-provider executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-jacc-provider,
- list-jacc-providers

Create-Javamail-Resource

Creates a JavaMail session resource.

Synopsis

```
asadmin [asadmin-options] create-javamail-resource [--help]
[--target target] --mailhost hostname
--mailuser username --fromaddress address [--storeprotocol storeprotocol]
[--storeprotocolclass storeprotocolclass] [--transprotocol transprotocol]
[--transprotocolclass transprotocolclass] [--debug={false|true}] [--enabled={
true|false}]
[--description resource-description] [--property (name=value)[:name=value]*] jndi-
name
```

Description

The `create-javamail-resource` subcommand creates a JavaMail session resource.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target for which you are creating the JavaMail session resource. Valid values are:
`server`;; Creates the resource for the default server instance. This is the default value. `domain`;; Creates the resource for the domain. cluster_name;; Creates the resource for every server instance in the cluster. instance_name;; Creates the resource for a particular server instance. **`--mailhost`**: The DNS name of the default mail server. The connect methods of the Store and Transport objects use this value if a protocol-specific host property is not supplied. The name must be resolvable to an actual host name. **`--mailuser`**: The name of the mail account user provided when connecting to a mail server. The connect methods of the Store and Transport objects use this value if a protocol-specific username property is not supplied. **`--fromaddress`**: The email address of the default user, in the form username`@`host`.`domain`.--storeprotocol`: The mail server store protocol. The default is `imap`. Change this value only if you have reconfigured the Payara Server's mail provider to use a non-default store protocol. **`--storeprotocolclass`**: The mail server store protocol class name. The default is `org.eclipse.angus.mail.imap.IMAPStore`. Change this value only if you have reconfigured the Payara Server's mail provider to use a nondefault store protocol. **`--transprotocol`**: The mail server transport protocol. The default is `smtp`. Change this value only if you have reconfigured the Payara Server's mail provider to use a nondefault transport protocol. **`--transprotocolclass`**: The mail server transport protocol class name. The default is `org.eclipse.angus.mail.smtp.SMTPTransport`. Change this value only if you have reconfigured the Payara Server's mail provider to use a nondefault transport

protocol. **`--debug`**: If set to true, the server starts up in debug mode for this resource. If the JavaMail log level is set to `FINE` or `FINER`, the debugging output will be generated and will be included in the server log file. The default value is false. **`--enabled`**: If set to true, the resource is enabled at runtime. The default value is true. **`--description`**: Text providing some details of the JavaMail resource. **`--property`**: Optional attribute name/value pairs for configuring the JavaMail resource. The Payara Server-specific `mail-` prefix is converted to the standard mail prefix. The JavaMail API documentation lists the properties you might want to set.

Operands

jndi-name: The JNDI name of the JavaMail resource to be created. It is a recommended practice to use the naming subcontext prefix `mail/` for JavaMail resources.

Examples

Example 1 Creating a JavaMail Resource

This example creates a JavaMail resource named `mail/MyMailSession`. The JNDI name for a JavaMail session resource customarily includes the `mail/` naming subcontext.

```
asadmin> create-javamail-resource --mailhost localhost
--mailuser sample --fromaddress sample@sun.com mail/MyMailSession
Command create-javamail-resource executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-javamail-resource,
- list-javamail-resources

Create-Jdbc-Connection-Pool

Registers a JDBC connection pool.

Synopsis

```
asadmin [asadmin-options] create-jdbc-connection-pool [--help]
[--datasourceclassname=datasourceclassname]
[--restype=resourcetype]
[--steadypoolsize=poolsize]
[--maxpoolsize=maxpoolsize]
[--maxwait=maxwaittime]
[--poolresize=poolresizelimit]
[--idletimeout=idletimeout]
[--initsql=initsqlstring]
[--isolationlevel=isolationlevel]
[--isisolationguaranteed={true|false}]
[--isconnectvalidatereq={false|true}]
[--validationmethod=validationmethod]
[--validationtable=validationtable]
[--failconnection={false|true}]
[--allownoncomponentcallers={false|true}]
[--nontransactionalconnections={false|true}]
[--validateatmostonceperiod=validationinterval]
[--leaktimeout=leaktimeout]
[--leakreclaim={false|true}]
[--statementleaktimeout=satementleaktimeout]
[--statmentleakreclaim={false|true}]
[--creationretryattempts=creationretryattempts]
[--creationretryinterval=creationretryinterval]
[--sqltracelisteners=sqltracelisteners[,sqltracelisteners]]
[--statementtimeout=statementtimeout]
[--lazyconnectionenlistment={false|true}]
[--lazyconnectionassociation={false|true}]
[--associatewiththread={false|true}]
[--driverclassname=jdbcdriverclassname]
[--matchconnections={false|true}]
[--maxconnectionusagecount=maxconnectionusagecount]
[--ping={false|true}]
[--pooling={false|true}]
[--statementcachesize=statementcachesize]
[--validationclassname=validationclassname]
[--wrapjdbcobjects={false|true}]
[--description description]
[--property name=value)[:name=value]*]
[--skipclientinfovalidation={false|true}]
connectionpoolid
```

Description

The `create-jdbc-connection-pool` subcommand registers a new Java Database Connectivity ("JDBC") software connection pool with the specified JDBC connection pool name.

A JDBC connection pool with authentication can be created either by using a `--property` option to specify user, password, or other connection information, or by specifying the connection information in the XML descriptor file.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--datasourceclassname`**: The name of the vendor-supplied JDBC datasource resource manager. An XA or global transactions capable datasource class will implement the `javax.sql.XADatasource` interface. Non-XA or exclusively local transaction datasources will implement the `javax.sql.Datasource` interface. `--restype`
Required when a datasource class implements two or more interfaces (`javax.sql.DataSource`, `javax.sql.XADataSource`, or `javax.sql.ConnectionPoolDataSource`), or when a driver classname must **be provided.**: * If `--restype` = `java.sql.Driver`, then the `--driverclassname` option is required. * If `--restype` = `javax.sql.DataSource`, `javax.sql.XADataSource`, or `javax.sql.ConnectionPoolDataSource`, then the `--datasourceclassname` option is required. * If `--restype` is not specified, then either the `--driverclassname` or `--datasourceclassname` option must be specified, but not both. **`--steadypoolsize`**: The minimum and initial number of connections maintained in the pool. The default value is 8. **`--maxpoolsize`**: The maximum number of connections that can be created. The default value is 32. **`--maxwait`**: The amount of time, in milliseconds, that a caller will wait before a connection timeout is sent. The default is 60000 (60 seconds). A value of 0 forces the caller to wait indefinitely. **`--poolresize`**: Number of connections to be removed when `idle-timeout-in-seconds` timer expires. This is the quantity by which the pool will scale up or scale down the number of connections. Scale up: When the pool has no free connections, pool will scale up by this quantity. Scale down: All the invalid and idle connections are removed, sometimes resulting in removing connections of quantity greater than this value. Connections that have been idle for longer than the timeout are candidates for removal. Steadypoolsize will be ensured. Possible values are from 0 to `MAX_INTEGER`. The default value is 2. **`--idletimeout`**: The maximum time, in seconds, that a connection can remain idle in the pool. After this time, the

implementation can close this connection. This timeout value must be kept shorter than the database server side timeout value to prevent the accumulation of unusable connections in the application. The default value is 300. **`--initsql`**: An SQL string that is executed whenever a connection is created from the pool. If an existing connection is reused, this string is not executed. Connections that have idled for longer than the timeout are candidates for removal. This option has no default value. **`--isolationlevel`**: The transaction-isolation-level on the pooled database connections. This option does not have a default value. If not specified, the pool operates with the default isolation level that the JDBC driver provides. You can set a desired isolation level using one of the standard transaction isolation levels: `read-uncommitted`, `read-committed`, `repeatable-read`, `serializable`. Applications that change the isolation level on a pooled connection programmatically risk polluting the pool. This could lead to program errors. **`--isisolationguaranteed`**: This is applicable only when a particular isolation level is specified for transaction-isolation-level. The default value is true. This option assures that every time a connection is obtained from the pool, isolation level is set to the desired value. This could have some performance impact on some JDBC drivers. Administrators can set this to false when the application does not change `--isolationlevel` before returning the connection. **`--isconnectvalidatereq`**: If set to true, connections are validated or checked to see if they are usable before giving out to the application. The default value is false. **`--validationmethod`**: Type of validation to be performed when `is-connection-validation-required` is true. Valid settings are: `auto-commit`, `meta-data`, `table`, or `custom-validation`. The default value is `table`. **`--validationtable`**: The name of the validation table used to perform a query to validate a connection. If `is-connection-validation-required` is set to true and connection-validation-type set to table, this option is mandatory. **`--failconnection`**: If set to true, all connections in the pool must be closed when a single validation check fails. The default value is false. One attempt is made to reestablish failed connections. **`--allownoncomponentcallers`**: A pool with this property set to true can be used by non-Jakarta EE components, that is, components other than EJBs or Servlets. The returned connection is enlisted automatically with the transaction context obtained

from the transaction manager. Connections obtained by non-component callers are not automatically cleaned by the container at the end of a transaction. These connections need to be explicitly closed by the caller. **`--nontransactionalconnections`**: A pool with this property set to true returns non-transactional connections. This connection does not get automatically enlisted with the transaction manager.
`--validateatmostonceperiod`: Specifies the time interval in seconds between successive requests to validate a connection at most once. Setting this attribute to an appropriate value minimizes the number of validation requests by a connection. Default value is 0, which means that the attribute is not enabled. **`--leaktimeout`**: Specifies the amount of time, in seconds, for which connection leaks in a connection pool are to be traced. When a connection is not returned to the pool by the application within the specified period, it is assumed to be a potential leak, and stack trace of the caller will be logged. This option only detects if there is a connection leak. The connection can be reclaimed only if `connection-leak-reclaim` is set to true. If connection leak tracing is enabled, you can use the Administration Console to enable monitoring of the JDBC connection pool to get statistics on the number of connection leaks. The default value is 0, which disables connection leak tracing. **`--leakreclaim`**: Specifies whether leaked connections are restored to the connection pool after leak connection tracing is complete. Possible values are as follows:
`false`;; Leaked connections are not restored to the connection pool (default). `true`;; Leaked connections are restored to the connection pool. **`--statementleaktimeout`**: Specifies the amount of time, in seconds, after which any statements that have not been closed by an application are to be detected. Applications can run out of cursors if statement objects are not properly closed. This option only detects if there is a statement leak. The statement can be reclaimed only if `statement-leak-reclaim` is set to true. The leaked statement is closed when it is reclaimed. The stack trace of the caller that creates the statement will be logged when a statement leak is detected. If statement leak tracing is enabled, you can use the Administration Console to enable monitoring of the JDBC connection pool to get statistics on the number of statement leaks. The default value is 0, which disables statement leak tracing. The following limitations apply to the statement leak timeout value:

+ * The value must be less than the value set for the connection `leak-timeout`. * The value must be greater than the value set for `statement-timeout`.
`--statementleakreclaim`: Specifies whether leaked statements are reclaimed after the statements leak. Possible values are as follows:
`false`;; Leaked statements are not reclaimed (default). `true`;; Leaked statements are reclaimed. **`--creationretryattempts`**: Specifies the maximum number of times that Payara retries to create a connection if the initial attempt fails. The default value is 0, which specifies that Payara does not retry to create the connection.
`--creationretryinterval`: Specifies the interval, in seconds, between successive attempts to create a connection. If `--creationretryattempts` is 0, the `--creationretryinterval` option is ignored. The default value is 10.
`--sqltracelisteners`: A list of one or more custom modules that provide custom logging of database activities. Each module must implement the `org.glassfish.api.jdbc.SQLTraceListener` public interface. When set to an appropriate value, SQL statements executed by applications are traced. This option has no default value. **`--statementtimeout`**: Specifies the length of time in seconds after which a query that is not completed is terminated. A query that remains incomplete for a long period of time might cause the application that submitted the query to hang. To prevent this occurrence, use this option set a timeout for all statements that will be created from the connection pool that you are creating. When creating a statement, Payara sets the `QueryTimeout` property on the statement to the length of time that is specified. The default value is -1, which specifies that incomplete queries are never terminated. **`--lazyconnectionenlistment`**: Specifies whether a resource to a transaction is enlisted only when a method actually uses the resource. Possible values are as follows:
`false`;; Resources to a transaction are always enlisted and not only when a method actually uses the resource (default). `true`;; Resources to a transaction are enlisted only when a method actually uses the resource. **`--lazyconnectionassociation`**: Specifies whether a physical connection should be associated with the logical connection only when the physical connection is used, and disassociated when the transaction is completed. Such association and dissociation enable the reuse of

physical connections. Possible values are as follows:
`false`;; A physical connection is associated with the logical connection even before the physical connection is used, and is not disassociated when the transaction is completed (default). `true`;; A physical connection is associated with the logical connection only when the physical connection is used, and disassociated when the transaction is completed. The `--lazyconnectionenlistment` option must also be set to `true`. **`--associatewiththread`**: Specifies whether a connection is associated with the thread to enable the thread to reuse the connection. If a connection is not associated with the thread, the thread must obtain a connection from the pool each time that the thread requires a connection. Possible values are as follows:
`false`;; A connection is not associated with the thread (default). `true`;; A connection is associated with the thread. **`--driverclassname`**: The name of the vendor-supplied JDBC driver class. This driver should implement the `java.sql.Driver` interface.
`--matchconnections`: Specifies whether a connection that is selected from the pool should be matched by the resource adaptor. If all the connections in the pool are homogenous, a connection picked from the pool need not be matched by the resource adapter, which means that this option can be set to false. Possible values are as follows:
`false`;; A connection should not be matched by the resource adaptor (default). `true`;; A connection should be matched by the resource adaptor.
`--maxconnectionusagecount`: Specifies the maximum number of times that a connection can be reused. When this limit is reached, the connection is closed. By limiting the maximum number of times that a connection can be reused, you can avoid statement leaks. The default value is 0, which specifies no limit on the number of times that a connection can be reused. **`--ping`**: Specifies if the pool is pinged during pool creation or reconfiguration to identify and warn of any erroneous values for its attributes. Default value is false. **`--pooling`**: Specifies if connection pooling is enabled for the pool. The default value is true. **`--statementcachesize`**: The number of SQL statements to be cached using the default caching mechanism (Least Recently Used). The default value is 0, which indicates that statement caching is not enabled.
`--validationclassname`: The name of the class that provides custom validation

when the value of `validationmethod` is `custom-validation`. This class must implement the `org.glassfish.api.jdbc.ConnectionValidation` interface, and it must be accessible to Payara. This option is mandatory if the connection validation type is set to custom validation. **`--wrapjdbcobjects`**: Specifies whether the pooling infrastructure provides wrapped JDBC objects to applications. By providing wrapped JDBC objects, the pooling infrastructure prevents connection leaks by ensuring that applications use logical connections from the connection pool, not physical connections. The use of logical connections ensures that the connections are returned to the connection pool when they are closed. However, the provision of wrapped JDBC objects can impair the performance of applications. The default value is true.
The pooling infrastructure provides wrapped objects for implementations of the following interfaces in the JDBC API:
* `java.sql.CallableStatement` * `java.sql.DatabaseMetaData` * `java.sql.PreparedStatement` * `java.sql.ResultSet` * `java.sql.Statement`
Possible values of `--wrapjdbcobjects` are as follows:
`false`;; The pooling infrastructure does not provide wrapped JDBC objects to applications. (default). `true`;; The pooling infrastructure provides wrapped JDBC objects to applications. **`--description`**: Text providing details about the specified JDBC connection pool. **`--property`**: Optional attribute name/value pairs for configuring the pool. The following properties are available:
`user`;; Specifies the username for connecting to the database. `password`;; Specifies the password for connecting to the database. `databaseName`;; Specifies the database for this connection pool. `serverName`;; Specifies the database server for this connection pool. `port`;; Specifies the port on which the database server listens for requests. `networkProtocol`;; Specifies the communication protocol. `roleName`;; Specifies the initial SQL role name. `datasourceName`;; Specifies an underlying `XADataSource`, or a `ConnectionPoolDataSource` if connection pooling is done. `description`;; Specifies a text description. `url`;; Specifies the URL for this connection pool. Although this is not a standard property, it is commonly used. `dynamic-reconfiguration-wait-timeout-in-seconds`;; Used to enable dynamic

reconfiguration of the connection pool transparently to the applications that are using the pool, so that applications need not be re-enabled for the attribute or property changes to the pool to take effect. Any in-flight transaction's connection requests will be allowed to complete with the old pool configuration as long as the connection requests are within the timeout period, so as to complete the transaction. New connection requests will wait for the pool reconfiguration to complete and connections will be acquired using the modified pool configuration. `Prefer-Validate-Over-Recreate`;; Specifies whether pool resizer should validate idle connections before destroying and recreating them. The default value is true. `time-to-keep-queries-in-minutes`;; Specifies the number of minutes that will be cached for use in calculating frequently used queries. Takes effect when SQL tracing and monitoring are enabled for the JDBC connection pool. The default value is 5 minutes. `number-of-top-queries-to-report`;; Specifies the number of queries to list when reporting the top and most frequently used queries. Takes effect when SQL tracing and monitoring are enabled for the JDBC connection pool. The default value is 10 queries.

NOTE	If an attribute name or attribute value contains a colon, the backslash (`\`) must be used to escape the colon in the name or value. Other characters might also require an escape character. For more information about escape characters in command options, see the asadmin-1m man page.

`--skipclientinfovalidation`: Determines whether to skip the validation of client info properties. Default is false.

NOTE	Skipping client info validation may improve the performance of persistent operations when pooling is disabled and managed by the JDBC driver.

Operands

connectionpoolid: The name of the JDBC connection pool to be created.

Examples

Example 1 Creating a JDBC Connection Pool

This example creates a JDBC connection pool named `sample_derby_pool`.

```
asadmin> create-jdbc-connection-pool
  --datasourceclassname org.h2.Driver
  --restype javax.sql.XADataSource
  --property portNumber=1527:password=APP:user=APP:serverName
=localhost:databaseName=sun-appserv-samples sample_h2_pool
Command create-jdbc-connection-pool executed successfully
```

The escape character backslash (`\`) is used in the `--property` option to distinguish the semicolon (`;`). Two backslashes (`\\`) are used to distinguish the equal sign (`=`).

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-jdbc-connection-pool,
- list-jdbc-connection-pools

Create-Jdbc-Resource

Creates a JDBC resource with the specified JNDI name.

Synopsis

```
asadmin [asadmin-options] create-jdbc-resource [--help]
--connectionpoolid connectionpoolid
[--enabled={false|true}]
[--description description]
[--property (property=value)[:name=value]*]
[--target target]
jndi_name
```

Description

The `create-jdbc-resource` subcommand creates a new JDBC resource.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options,

see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--connectionpoolid`: The name of the JDBC connection pool. If two or more JDBC resource elements point to the same connection pool element, they use the same pool connection at runtime. `--enabled`: Determines whether the JDBC resource is enabled at runtime. The default value is true. `--description`: Text providing descriptive details about the JDBC resource. `--property`: Optional attribute name/value pairs for configuring the resource. `--target`: This option helps specify the target to which you are deploying. Valid values are:
`server`;; Deploys the component to the default server instance. This is the default value. `domain`;; Deploys the component to the domain. cluster_name;; Deploys the component to every server instance in the cluster. instance_name;; Deploys the component to a particular server instance.

NOTE

The resource is always created for the domain as a whole, but the `resource-ref` for the resource is only created for the specified `--target`. This means that although the resource is defined at the domain level, it is only available at the specified target level. Use the `create-resource-ref` subcommand to refer to the resource in multiple targets if needed.

Operands

jndi_name: The JNDI name of this JDBC resource.

Examples

Example 1 Creating a JDBC Resource

This example creates a JDBC resource named `jdbc/DerbyPool`.

```
asadmin> create-jdbc-resource
--connectionpoolid sample_h2_pool jdbc/H2Pool
Command create-jdbc-resource executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-resource-ref,
- delete-jdbc-resource,
- list-jdbc-resources

Create-Jms-Host

Creates a JMS host.

Synopsis

```
asadmin [asadmin-options] create-jms-host [--help]
--mqhost mq-host --mqport mq-port
--mquser mq-user --mqpassword mq-password
[--target target]
[--force={false|true}]
jms_host_name
```

Description

Creates a Java Message Service (JMS) host within the JMS service.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--mqhost`**: The host name for the JMS service. **`--mqport`**: The port number used by the JMS service. **`--mquser`**: The user name for the JMS service. **`--mqpassword`**: The password for the JMS service. **`--target`**: Creates the JMS host only for the specified target. Valid values are as follows:
`server`;; Creates the JMS host for the default server instance. This is the default value. configuration-name;; Creates the JMS host in the specified configuration. cluster-name;; Creates the JMS host for every server instance in the specified cluster. instance-name;; Creates the JMS host for the specified server instance. **`--force`**: Specifies whether the

subcommand overwrites the existing JMS host of the same name. The default value is `false`.

Operands

jms_host_name: A unique identifier for the JMS host to be created.

Examples

Example 1 Creating a JMS host using a non-default port

The following command creates a JMS host named `MyNewHost` on the system `pigeon`.

```
asadmin> create-jms-host --mqhost pigeon.example.com --mqport 7677
--mquser admin --mqpassword admin MyNewHost
Jms Host MyNewHost created.
Command create-jms-host executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-jms-host,
- jms-ping,
- list-jms-hosts

Create-Jms-Resource

Creates a JMS resource.

Synopsis

```
asadmin [asadmin-options] create-jms-resource [--help]
--restype type
[--target target]
[--enabled={true|false}]
[--description text]
[--property (name=value)[:name=value]*]
[--force={false|true}]
jndi_name
```

Description

The `create-jms-resource` subcommand creates a Java Message Service (JMS) connection factory resource or a JMS destination resource.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--restype`**: The JMS resource type, which can be `jakarta.jms.Topic`, `jakarta.jms.Queue`, `jakarta.jms.ConnectionFactory`, `jakarta.jms.TopicConnectionFactory`, or `jakarta.jms.QueueConnectionFactory`. **`--target`**: Creates the JMS resource only for the specified target. Valid values are as follows:

NOTE	The resource is always created for the domain as a whole, but the`<resource-ref>` element for the resource is only created for the specified `--target`. This means that although the resource is defined at the domain level, it is only active at the specified `--target`.

`server`

Creates the JMS resource for the default server instance. This is the default value.

`domain`

Creates the JMS resource for the domain.

cluster-name

Creates the JMS resource for every server instance in the specified cluster.

instance-name

Creates the JMS resource for the specified server instance. **`--enabled`**: If set to true (the default), the resource is enabled at runtime. **`--description`**: Text

providing details about the JMS resource. `--property`: Optional attribute name/value pairs for configuring the JMS resource. You can specify the following properties for a connection factory resource:

`ClientId`

A client ID for a connection factory that will be used by a durable subscriber.

`AddressList`

A comma-separated list of message queue addresses that specify the host names (and, optionally, port numbers) of a message broker instance or instances with which your application will communicate. For example, the value could be `earth` or `earth:7677`. Specify the port number if the message broker is running on a port other than the default (7676). The default value is an address list composed of the JMS hosts defined in the server's JMS service configuration. The default value is `localhost` and the default port number is 7676. The client will attempt a connection to a broker on port 7676 of the local host.

`UserName`

The username for the connection factory. The default value is `guest`.

`Password`

The password for the connection factory. The default value is `guest`.

`ReconnectEnabled`

A value of `true` indicates that the client runtime attempts to reconnect to a message server (or the list of addresses in the `AddressList`) when a connection is lost. The default value is `false`.

`ReconnectAttempts`

The number of attempts to connect (or reconnect) for each address in the `AddressList` before the client runtime tries the next address in the list. A value of -1 indicates that the number of reconnect attempts is unlimited (the client runtime attempts to connect to the first address until it succeeds). The default

value is 6.

`ReconnectInterval`

The interval in milliseconds between reconnect attempts. This applies to attempts on each address in the `AddressList` and for successive addresses in the list. If the interval is too short, the broker does not have time to recover. If it is too long, the reconnect might represent an unacceptable delay. The default value is 30,000 milliseconds.

`AddressListBehavior`

Specifies whether connection attempts are in the order of addresses in the `AddressList` (`PRIORITY`) or in a random order (`RANDOM`). `PRIORITY` means that the reconnect will always try to connect to the first server address in the `AddressList` and will use another one only if the first broker is not available. If you have many clients attempting a connection using the same connection factory, specify `RANDOM` to prevent them from all being connected to the same address. The default value is the `AddressListBehavior` value of the server's JMS service configuration.

`AddressListIterations`

The number of times the client runtime iterates through the `AddressList` in an effort to establish (or re-establish) a connection). A value of -1 indicates that the number of attempts is unlimited. The default value is -1.
Additionally, you can specify `connector-connection-pool` attributes as connector resource properties. For a list of these attributes, see "connector-connection-pool" in Payara Server Application Deployment section.
You can specify the following properties for a destination resource:

`Name`

The name of the physical destination to which the resource will refer. The physical destination is created automatically when you run an application that uses the destination resource. You can also create a physical destination with the `create-`

`jmsdest` subcommand. If you do not specify this property, the JMS service creates a physical destination with the same name as the destination resource (replacing any forward slash in the JNDI name with an underscore).

`Description`

A description of the physical destination. **`--force`**: Specifies whether the subcommand overwrites the existing JMS resource of the same name. The default value is `false`.

Operands

jndi_name: The JNDI name of the JMS resource to be created.

Examples

Example 1 Creating a JMS connection factory resource for durable subscriptions

The following subcommand creates a connection factory resource of type `jakarta.jms.ConnectionFactory` whose JNDI name is `jms/DurableConnectionFactory`. The `ClientId` property sets a client ID on the connection factory so that it can be used for durable subscriptions. The JNDI name for a JMS resource customarily includes the `jms/` naming subcontext.

```
asadmin> create-jms-resource --restype jakarta.jms.ConnectionFactory
--description "connection factory for durable subscriptions"
--property ClientId=MyID jms/DurableConnectionFactory
Connector resource jms/DurableConnectionFactory created.
Command create-jms-resource executed successfully.
```

Example 2 Creating a JMS destination resource

The following subcommand creates a destination resource whose JNDI name is `jms/MyQueue`. The `Name` property specifies the physical destination to which the resource refers.

```
asadmin> create-jms-resource --restype jakarta.jms.Queue
--property Name=PhysicalQueue jms/MyQueue
Administered object jms/MyQueue created.
Command create-jms-resource executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-jms-resource,
- list-jms-resources

Create-Jmsdest

Creates a JMS physical destination.

Synopsis

```
asadmin [asadmin-options] create-jmsdest [--help]
--desttype dest_type
[--property (name=value)[:name=value]*]
[--target target]
[--force={false|true}]
dest_name
```

Description

The `create-jmsdest` subcommand creates a Java Message Service (JMS) physical destination. Typically, you use the `create-jms-resource` subcommand to create a JMS destination resource that has a `Name` property that specifies the physical destination. The physical destination is created automatically when you run an application that uses the destination resource. Use the `create-jmsdest` subcommand if you want to create a physical destination with non-default property settings.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--desttype`**: The type of the JMS destination. Valid values are `topic`

and `queue`. **`--property`**: Optional attribute name/value pairs for configuring the physical destination. You can specify the following properties for a physical destination. `MaxNumMsgs`;; The maximum number of unconsumed messages permitted for the destination. A value of -1 denotes an unlimited number of messages. The default value is -1. For the dead message queue, the default value is 1000. If the `limitBehavior` property is set to `FLOW_CONTROL`, it is possible for the specified message limit to be exceeded because the broker cannot react quickly enough to stop the flow of incoming messages. In such cases, the value specified for `maxNumMsgs` serves as merely a hint for the broker rather than a strictly enforced limit. `MaxBytesPerMsg`;; The maximum size, in bytes, of any single message. Rejection of a persistent message is reported to the producing client with an exception; no notification is sent for non-persistent messages.

The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes:

`b`: Bytes **`k`**: Kilobytes (1024 bytes) **`m`**: Megabytes (1024 x 1024 = 1,048,576 bytes) A value with no suffix is expressed in bytes; a value of -1 denotes an unlimited message size. The default value is -1.

`MaxTotalMsgBytes`

The maximum total memory, in bytes, for unconsumed messages. The default value is -1. The syntax is the same as for `maxBytesPerMsg`. For the dead message queue, the default value is `10m`.

`LimitBehavior`

The behavior of the message queue broker when the memory-limit threshold is reached.+ **Valid values are as follows.**: **`REJECT_NEWEST`**: Reject newest messages and notify the producing client with an exception only if the message is persistent. This is the default value. **`FLOW_CONTROL`**: Slow the rate at which message producers send messages. **`REMOVE_OLDEST`**: Throw out the oldest messages. **`REMOVE_LOW_PRIORITY`**: Throw out the lowest-priority messages according to age, with no notification to the producing

client. If the value is `REMOVE_OLDEST` or `REMOVE_LOW_PRIORITY` and the `useDMQ` property is set to `true`, excess messages are moved to the dead message queue. For the dead message queue itself, the default limit behavior is `REMOVE_OLDEST`, and the value cannot be set to `FLOW_CONTROL`.

`MaxNumProducers`

The maximum number of message producers for the destination. When this limit is reached, no new producers can be created. A value of -1 denotes an unlimited number of producers. The default value is 100. This property does not apply to the dead message queue.

`ConsumerFlowLimit`

The maximum number of messages that can be delivered to a consumer in a single batch. A value of -1 denotes an unlimited number of messages. The default value is 1000. The client runtime can override this limit by specifying a lower value on the connection factory object.
In load-balanced queue delivery, this is the initial number of queued messages routed to active consumers before load balancing begins.

`UseDMQ`

If set to `true`, dead messages go to the dead message queue. If set to `false`, dead messages are discarded. The default value is `true`.

`ValidateXMLSchemaEnabled`

If set to `true`, XML schema validation is enabled for the destination. The default value is `false`.
When XML validation is enabled, the Message Queue client runtime will attempt to validate an XML message against the specified XSDs (or against the DTD, if no XSD is specified) before sending it to the broker. If the specified schema cannot be located or the message cannot be validated, the message is not sent, and an exception is thrown.

This property should be set when a destination is inactive: that is, when it has no consumers or producers and when there are no messages in the destination. Otherwise, the producer must reconnect.

`XMLSchemaURIList`

A space-separated list of XML schema document (XSD) URI strings. The URIs point to the location of one or more XSDs to use for XML schema validation, if `validateXMLSchemaEnabled` is set to `true`. The default value is `null`. Use double quotes around this value if multiple URIs are specified, as in the following example:

```
"http://foo/flap.xsd http://test.com/test.xsd"
```

If this property is not set or `null` and XML validation is enabled, XML validation is performed using a DTD specified in the XML document. If an XSD is changed as a result of changing application requirements, all client applications that produce XML messages based on the changed XSD must reconnect to the broker.
To modify the value of these properties, you can use the `as-install/mq/bin/imqcmd` command. See the **Physical Destination Property Reference** in the Open Message Queue Administration Guide for more information.
`--target`: Creates the physical destination only for the specified target. Although the `create-jmsdest` subcommand is related to resources, a physical destination is created using the JMS Service (JMS Broker), which is part of the configuration. A JMS Broker is configured in the config section of `domain.xml`. Valid values are as follows: `server`;; Creates the physical destination for the default server instance. This is the default value. configuration-name;; Creates the physical destination in the specified configuration. cluster-name;; Creates the physical destination for every server instance in the specified cluster. instance-name;; Creates the physical destination for the specified server instance. **`--force`**: Specifies whether the subcommand overwrites the existing JMS physical destination of the same name. The default value is `false`.

Operands

dest_name: A unique identifier for the JMS destination to be created.

Examples

Example 1 Creating a JMS physical destination

The following subcommand creates a JMS physical queue named `PhysicalQueue` with non-default property values.

```
asadmin> create-jmsdest --desttype queue
--property maxNumMsgs=1000:maxBytesPerMsg=5k PhysicalQueue
Command create-jmsdest executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-jms-resource,
- delete-jmsdest,
- flush-jmsdest,
- list-jmsdest

Create-Jndi-Resource

Registers a JNDI resource.

Synopsis

```
asadmin [asadmin-options] create-jndi-resource [--help]
[--target target]
--restype restype --factoryclass factoryclass
--jndilookupname jndilookupname [--enabled={true|false}]
[--description description]
[--property (name=value)[:name=value]*]
jndi-name
```

Description

The `create-jndi-resource` subcommand registers a JNDI resource.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target for which you are registering a JNDI resource. Valid values for target are described below.

> **NOTE**
>
> The resource is always created for the domain as a whole, but the `resource-ref` for the resource is only created for the specified `--target`. This means that although the resource is defined at the domain level, it is only available at the specified target level. Use the `create-resource-ref` subcommand to refer to the resource in multiple targets if needed.

`server`

Creates the resource for the default server instance. This value is the default.

`domain`

Creates the resource for the domain

cluster-name

Creates the resource for every server instance in the cluster

instance-name

Creates the resource for a particular server instance **`--restype`**: The JNDI resource type. Valid values are `topic` or `queue`. **`--factoryclass`**: The class that creates the JNDI resource. **`--jndilookupname`**: The lookup name that the external container uses. **`--enabled`**: Determines whether the resource is enabled at runtime. Default is true. **`--description`**: The text that provides details about the JNDI resource. **`--property`**: Optional properties for configuring the resource. Each property is specified as a name-value pair. The available properties are specific to the implementation that is specified by the `--factoryclass` option

and are used by that implementation. Payara Server itself does not define any properties for configuring a JNDI resource.

Operands

jndi-name: The unique name of the JNDI resource to be created.

Examples

Example 1 Creating a JNDI Resource

This example creates the JNDI resource `my-jndi-resource` for the default server instance.

```
asadmin> create-jndi-resource
--restype com.example.jndi.MyResourceType
--factoryclass com.example.jndi.MyInitialContextFactoryClass
--jndilookupname remote-jndi-name
--description "sample JNDI resource" my-jndi-resource
JNDI resource my-jndi-resource created.
Command create-jndi-resource executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-resource-ref,
- delete-jndi-resource,
- list-jndi-resources

Create-Jvm-Options

Create options for the Java application launcher.

Synopsis

```
asadmin [asadmin-options] create-jvm-option [--help]
[--target target]
(assignment=jvm-option)
```

Description

The `create-jvm-options` subcommand creates a single command-line option that is passed to the Java application launcher when Payara Server is started. The options that this subcommand creates are in addition to the options that are preset with Payara Server. Java application launcher options are stored in the Java configuration `java-config` element of the `domain.xml` file. The options are sent to the command line in the order they appear in the `java-config` element in the `domain.xml` file.

This subcommand can be used to create the following types of options:

- Java system properties. These options are set through the `-D` option of the Java application launcher. For example:
  `-Djava.security.manager`
  `-Denvironment=Production`
- Startup parameters for the Java application launcher. These options are preceded by the dash character (`-`). For example:
  `--XX:PermSize=\`size`
  `` `-Xmx1024m``
  `-d64`

If the subcommand specifies an option that already exists, the command does not re-create it.

NOTE	Ensure that any option that you create is valid. The subcommand might allow you to create an invalid option, but such an invalid option can cause startup to fail.

An option can be verified by examining the server log after Payara Server starts. Options for the Java application launcher are written to the `server.log` file before any other information when Payara Server starts.

The addition of some options requires a server restart for changes to become effective. Other options are set immediately in the environment of the domain administration

server (DAS) and do not require a restart. Whether a restart is required depends on the type of option.

- Restart is not required for Java system properties whose names do not start with `-Djava.` or `-Djavax.` (including the trailing period). For example, restart is not required for the following Java system property:
  `-Denvironment=Production`
- Restart is required for the following options:
  - Java system properties whose names start with `-Djava.` or `-Djavax.` (including the trailing period). For example:
    `-Djava.security.manager`
  - Startup parameters for the Java application launcher. For example:
    `-client`
    `-Xmx1024m`
    `-d64`

To restart the DAS, use the restart-domain command.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target on which you are creating Java application launcher options.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Operands

jvm-option-name: A single option where the format depends on the following:
* If the option has a name and a value, the format is option-name=value. * If the option

has only a name, the format is option-name. For example, `-Xmx2048m`. * If the first option name could be misinterpreted as one or more `asadmin` short options, the format is `--` option-name. For example, `-server` in the following command could be misinterpreted as `-se`, the `asadmin` short forms for `--secure` and `--echo`:

+

```
create-jvm-option -server
```

To create the JVM option `-server`, instead use the command:

+

```
create-jvm-option -- -server
```

Examples

Example 1 Setting A Java System Property

```
asadmin> create-jvm-option -Dunixlocation=/root/example
Successfully created JVM option: -Dunixlocation=/root/example
Command create-jvm-option executed successfully.
```

Example 2 Setting a Startup Parameter for the Java Application Launcher

This example sets the maximum available heap size to 1024.

```
asadmin> create-jvm-option -Xmx1024m
Successfully added JVM option: -Xmx1024m
Command create-jvm-option executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jvm-options,

- delete-jvm-options,
- list-jvm-options,
- restart-domain

For more information about the Java application launcher, see the reference page for the operating system that you are using:

- https://docs.oracle.com/en/java/javase/11/ [Windows: java - the Java application launcher] (`https://docs.oracle.com/en/java/javase/11/`)

Create-Jvm-Options

Create options for the Java application launcher.

Synopsis

```
asadmin [asadmin-options] create-jvm-options [--help]
[--target target]
(jvm-option-name=jvm-option-value) [:jvm-option-name=jvm-option-value*]
```

Description

The `create-jvm-options` subcommand creates command-line options that are passed to the Java application launcher when Payara Server is started. The options that this subcommand creates are in addition to the options that are preset with Payara Server. Java application launcher options are stored in the Java configuration `java-config` element of the `domain.xml` file. The options are sent to the command line in the order they appear in the `java-config` element in the `domain.xml` file.

This subcommand can be used to create the following types of options:

- Java system properties. These options are set through the `-D` option of the Java application launcher. For example:
  `-Djava.security.manager`
  `-Denvironment=Production`
- Startup parameters for the Java application launcher. These options are preceded by the dash character (`-`). For example:

`--XX:PermSize=`size`

`-Xmx1024m`

`-d64`

If the subcommand specifies an option that already exists, the command does not re-create it.

NOTE	Ensure that any option that you create is valid. The subcommand might allow you to create an invalid option, but such an invalid option can cause startup to fail.

An option can be verified by examining the server log after Payara Server starts. Options for the Java application launcher are written to the `server.log` file before any other information when Payara Server starts.

The addition of some options requires a server restart for changes to become effective. Other options are set immediately in the environment of the domain administration server (DAS) and do not require a restart. Whether a restart is required depends on the type of option.

- Restart is not required for Java system properties whose names do not start with `-Djava.` or `-Djavax.` (including the trailing period). For example, restart is not required for the following Java system property:
  `-Denvironment=Production`
- Restart is required for the following options:
  - Java system properties whose names start with `-Djava.` or `-Djavax.` (including the trailing period). For example:
    `-Djava.security.manager`
  - Startup parameters for the Java application launcher. For example:
    `-client`
    `-Xmx1024m`
    `-d64`

To restart the DAS, use the restart-domain command.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target on which you are creating Java application launcher options.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Operands

jvm-option-name: One or more options delimited by a colon (:). The format of an option depends on the following:
* If the option has a name and a value, the format is option-name=value. * If the option has only a name, the format is option-name. For example, `-Xmx2048m`. * If the first option name could be misinterpreted as one or more `asadmin` short options, the format is `--` option-name. For example, `-server` in the following command could be misinterpreted as `-se`, the `asadmin` short forms for `--secure` and `--echo`:

+

```
create-jvm-options -server
```

To create the JVM option `-server`, instead use the command:

+

```
create-jvm-options -- -server
```

NOTE	If an option name or option value contains a colon, the backslash (`\`)

> must be used to escape the colon in the name or value. Other characters might also require an escape character. For more information about escape characters in subcommand options, see the asadmin-1m man page.

Examples

Example 1 Setting Java System Properties

This example sets multiple Java system properties.

```
asadmin> create-jvm-options -Dunixlocation=/root/example:
-Dvariable=\$HOME:-Dwindowslocation=d\:\\sun\\appserver:-Doption1=-value1
created 4 option(s)
Command create-jvm-options executed successfully.
```

Example 2 Setting a Startup Parameter for the Java Application Launcher

This example sets the maximum available heap size to 1024.

```
asadmin> create-jvm-options -Xmx1024m
created 1 option(s)
Command create-jvm-options executed successfully.
```

Example 3 Setting Multiple Startup Parameters for the Java Application Launcher

This example sets the maximum available heap size to 1024 and requests details about garbage collection.

```
asadmin> create-jvm-options "-Xmx1024m:-XX\:+PrintGCDetails"
created 1 option(s)
Command create-jvm-options executed successfully.
```

In this case, one of the two parameters already exists, so the subcommand reports that only one option was set.

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jvm-option,
- delete-jvm-options,
- list-jvm-options,
- restart-domain

For more information about the Java application launcher, see the reference page for the operating system that you are using:

- https://docs.oracle.com/en/java/javase/11/ [Windows: java - the Java application launcher] (`https://docs.oracle.com/en/java/javase/11/`)

Create-Local-Instance

Creates a Payara Server instance on the host where the subcommand is run.

Synopsis

```
asadmin [asadmin-options] create-local-instance [--help]
[--node node-name] [--nodedir node-dir]
[--config config-name | --cluster cluster-name]
[--lbenabled={true|false}]
[--portbase port-number] [--checkports={true|false}]
[--savemasterpassword={false|true}]
[--masterpasswordlocation filepath]
[--usemasterpassword={false|true}]
[--systemproperties (name=value)[:name=value]* ]
[--dataGridStartPort startPort]
instance-name
```

Description

The `create-local-instance` subcommand creates a Payara Server instance on the node that represents the host where the subcommand is run. This subcommand does not require the secure shell (SSH) to be configured.

You must run this subcommand from the host that is represented by the node where the instance is to reside. To contact the domain administration server (DAS), this subcommand requires the name of the host where the DAS is running.

If a non-default port is used for administration, this subcommand also requires the port

number. If you are adding the first instance to a node, you must provide this information through the `--host` option and the `--port` option of the asadmin-1m utility. For the second and later instances, this information is obtained from the DAS properties of the node.

A Payara Server instance is a single Virtual Machine for the Java platform (Java Virtual Machine or JVM machine) on a single node in which Payara Server is running. A node defines the host where the Payara Server instance resides. The JVM machine must be compatible with the Java Platform, Enterprise Edition (Jakarta EE).

A Payara Server instance requires a reference to the following items:

- The node that defines the host where the instance resides. The node can be specified in the command to create the instance, but is required only if more than one node exists in the directory where files for nodes are stored. If no node is specified, the behavior of the subcommand depends on the number of existing nodes in the directory where nodes are stored:
  - If no nodes exist, the subcommand creates a node for the instance. The name of the node is the name of the host on which the subcommand is run.
  - If only one node exists, the subcommand creates a reference to the existing node for the instance.
  - If two or more nodes exist, an error occurs.
- The named configuration that defines the configuration of the instance. The configuration can be specified in the command to create the instance, but is not required. If no configuration is specified for an instance that is not joining a cluster, the subcommand creates a configuration for the instance. An instance that is joining a cluster receives its configuration from its parent cluster.

Each Payara Server instance is one of the following types of instance:

Standalone instance: A standalone instance does not share its configuration with any other instances or clusters. A standalone instance is created if either of the following

conditions is met:
* No configuration or cluster is specified in the command to create the instance. * A configuration that is not referenced by any other instances or clusters is specified in the command to create the instance.

+ When no configuration or cluster is specified, a copy of the `default-config` configuration is created for the instance. The name of this configuration is instance-name`-config`, where instance-name represents the name of an unclustered server instance. **Shared instance**: A shared instance shares its configuration with other instances or clusters. A shared instance is created if a configuration that is referenced by other instances or clusters is specified in the command to create the instance. **Clustered instance**: A clustered instance inherits its configuration from the cluster to which the instance belongs and shares its configuration with other instances in the cluster. A clustered instance is created if a cluster is specified in the command to create the instance.

+ Any instance that is not part of a cluster is considered an unclustered server instance. Therefore, standalone instances and shared instances are unclustered server instances.

By default, this subcommand attempts to resolve possible port conflicts for the instance that is being created. The subcommand also assigns ports that are currently not in use and not already assigned to other instances on the same node.

The subcommand assigns these ports on the basis of an algorithm that is internal to the subcommand. Use the `--systemproperties` option to resolve port conflicts for additional instances on the same node. System properties of an instance can be manipulated by using the create-system-properties subcommand and the delete-system-property subcommand.

When creating an instance, the subcommand retrieves the files that are required for secure synchronization with the domain administration server (DAS). The instance is synchronized with the DAS when the instance is started

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--node`**: The name of the node that defines the host where the instance is to be created. The node must be specified only if more than one node exists in the directory where nodes are stored. Otherwise, the node may be omitted. If a node is specified, the node must exist. If no node is specified, the behavior of the subcommand depends on the number of existing nodes in the directory where nodes are stored:

+ * If no nodes exist, the subcommand creates a node for the instance. The name of the node is the name of the host on which the subcommand is run. * If only one node exists, the subcommand creates a reference to the existing node for the instance. * If two or more nodes exist, an error occurs. **`--nodedir`**: The path to the directory in which the files for instance's node is to be stored. The default is as-install`/nodes`. **`--config`**: Specifies the named configuration that the instance references. The configuration must exist and must not be named `default-config` or `server-config`. Specifying the `--config` option creates a shared instance. The `--config` option and the `--cluster` option are mutually exclusive. If both options are omitted, a standalone instance is created. **`--cluster`**: Specifies the cluster from which the instance inherits its configuration. Specifying the `--cluster` option creates a clustered instance. The `--config` option and the `--cluster` option are mutually exclusive. If both options are omitted, a standalone instance is created. **`--lbenabled`**: Specifies whether the instance is enabled for load balancing. Possible values are as follows: `true`;; The instance is enabled for load balancing (default). When an instance is enabled for load balancing, a load balancer sends requests to the instance. `false`;; The instance is disabled for load balancing. When an instance is disabled for load balancing, a load balancer does not send requests to the instance. **`--portbase`**: Determines the number with which the port assignment should start. An instance uses a certain number of ports that are statically assigned. The `portbase` value determines where the assignment should start. The values for the ports are calculated as follows: * Administration port: `portbase + 48` * HTTP listener port: `portbase + 80` * HTTPS listener port: `portbase + 81` * JMS port: `portbase + 76` * IIOP listener port:

`portbase + 37` * Secure IIOP listener port: `portbase + 38` * Secure IIOP with mutual authentication port: `portbase + 39` * JMX port: `portbase + 86` * JPA debugger port: `portbase + 9` * Felix shell service port for OSGi module management: `portbase + 66`

+ When the `--portbase` option is specified, the output of this subcommand includes a complete list of used ports. **`--checkports`**: Specifies whether to check for the availability of the administration, HTTP, JMS, JMX, and IIOP ports. The default value is `true`. **`--savemasterpassword`**: Setting this option to `true` allows the master password to be written to the file system. If the master password is written to the file system, the instance can be started without the need to prompt for the password. If this option is `true`, the `--usemasterpassword` option is also true, regardless of the value that is specified on the command line. Because writing the master password to the file system is an insecure practice, the default is `false`.
The master-password file for an instance is saved in the node directory, not the domain directory. Therefore, this option is required only for the first instance that is created for each node in a domain. **`--masterpasswordlocation`**: Only applies if the `--savemasterpassword` parameter is set to true. This parameter receives an absolute filepath that the master password will be saved to.
If this parameter is omitted, the default location will be used. **`--usemasterpassword`**: Specifies whether the key store is encrypted with a master password that is built into the system or a user-defined master password.
If `false` (default), the keystore is encrypted with a well-known password that is built into the system. Encrypting the keystore with a password that is built into the system provides no additional security.
If `true`, the subcommand obtains the master password from the `AS_ADMIN_MASTERPASSWORD` entry in the password file or prompts for the master password. The password file is specified in the `--passwordfile` option of the asadmin-1mutility.
If the `--savemasterpassword` option is `true`, this option is also true, regardless of the value that is specified on the command line.
The master password must be the same for all instances in a domain.

`--systemproperties`: Defines system properties for the instance. These properties override property definitions for port settings in the instance's configuration. Predefined port settings must be overridden if, for example, two clustered instances reside on the same host. In this situation, port settings for one instance must be overridden because both instances share the same configuration.
The following properties are available:
`ASADMIN_LISTENER_PORT`;; This property specifies the port number of the HTTP port or HTTPS port through which the DAS connects to the instance to manage the instance. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `HTTP_LISTENER_PORT`;; This property specifies the port number of the port that is used to listen for HTTP requests. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges.
`HTTP_SSL_LISTENER_PORT`;; This property specifies the port number of the port that is used to listen for HTTPS requests. Valid value are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `IIOP_LISTENER_PORT`;; This property specifies the port number of the port that is used for IIOP connections. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `IIOP_SSL_LISTENER_PORT`;; This property specifies the port number of the port that is used for secure IIOP connections. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges.
`IIOP_SSL_MUTUALAUTH_PORT`;; This property specifies the port number of the port that is used for secure IIOP connections with client authentication. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `JAVA_DEBUGGER_PORT`;; This property specifies the port number of the port that is used for connections to the Java Platform Debugger Architecture (JPDA) (https://docs.oracle.com/en/java/javase/11/docs/specs/jpda/jpda.html) debugger. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `JMS_PROVIDER_PORT`;; This property specifies the port number for the Java Message Service provider. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges.
`JMX_SYSTEM_CONNECTOR_PORT`;; This property specifies the port number on which

the JMX connector listens. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. `OSGI_SHELL_TELNET_PORT`;; This property specifies the port number of the port that is used for connections to the Apache Felix Remote Shell (`http://felix.apache.org/site/apache-felix-remote-shell.html`). This shell uses the Felix shell service to interact with the OSGi module management subsystem. Valid values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. **`--dataGridStartPort`**: Sets Data Grid (Hazelcast) instance's starting port.

+ If set to `0` the Domain wide start port will be used instead.

+ The default value is `0`.

Operands

instance-name: The name of the instance that is being created.

+ The name must meet the following requirements:

+ * The name may contain only ASCII characters. * The name must start with a letter, a number, or an underscore. * The name may contain only the following characters: **Lowercase letters** Uppercase letters **Numbers** Hyphen **Period** Underscore * The name must be unique in the domain and must not be the name of another Payara Server instance, a cluster, a named configuration, or a node. * The name must not be `domain`, `server`, or any other keyword that is reserved by Payara Server.

Examples

Example 1 Creating a Standalone Payara Server Instance

This example creates the standalone instance `il3` on the host where the command is run. The DAS is running on the same host. The instance references the only existing node.

```
asadmin> create-local-instance il3
Rendezvoused with DAS on localhost:4848.
Port Assignments for server instance il3:
JMX_SYSTEM_CONNECTOR_PORT=28686
JMS_PROVIDER_PORT=27676
```

```
HTTP_LISTENER_PORT=28080
ASADMIN_LISTENER_PORT=24848
JAVA_DEBUGGER_PORT=29009
IIOP_SSL_LISTENER_PORT=23820
IIOP_LISTENER_PORT=23700
OSGI_SHELL_TELNET_PORT=26666
HTTP_SSL_LISTENER_PORT=28181
IIOP_SSL_MUTUALAUTH_PORT=23920
Command create-local-instance executed successfully.
```

Example 2 Creating a Clustered Payara Server Instance on a Specific Node

This example creates the clustered instance `ymli2` on node `sj02`. The instance is a member of the cluster `ymlclust`.

The command is run on the host `sj02`, which is the host that the node `sj02` represents. The DAS is running on the host `sr04` and uses the default HTTP port for administration. Because no instances exist on the node, the host on which the DAS is running is provided through the `--host` option of the `asadmin` utility.

```
sj02# asadmin --host sr04 create-local-instance --cluster ymlclust --node sj02
ymli2
Rendezvoused with DAS on sr04:4848.
Port Assignments for server instance ymli2:
JMX_SYSTEM_CONNECTOR_PORT=28686
JMS_PROVIDER_PORT=27676
HTTP_LISTENER_PORT=28080
ASADMIN_LISTENER_PORT=24848
JAVA_DEBUGGER_PORT=29009
IIOP_SSL_LISTENER_PORT=23820
IIOP_LISTENER_PORT=23700
OSGI_SHELL_TELNET_PORT=26666
HTTP_SSL_LISTENER_PORT=28181
IIOP_SSL_MUTUALAUTH_PORT=23920
Command create-local-instance executed successfully.
```

Example 3 Creating a Standalone Payara Server Instance with Specific Data Grid Start Port

This example creates a Standalone Payara instance `instance2` in the domain `domain1` on the local host. Setting the Data Grid Start Port when an instance is created to `2900`.

```
asadmin> create-local-instance --dataGridStartPort 2900 --node localhost-domain1
instance2
```

```
Rendezvoused with DAS on localhost:4848.
Using DAS host localhost and port 4848 from existing das.properties for node
localhost-domain1. To use a different DAS, create a new node using create-node-ssh
or
create-node-config. Create the instance with the new node and correct
host and port:
asadmin --host das_host --port das_port create-local-instance --node node_name
instance_name.
Port Assignments for server instance instance2:
OSGI_SHELL_TELNET_PORT=26667
JAVA_DEBUGGER_PORT=29010
JMS_PROVIDER_PORT=27677
HTTP_LISTENER_PORT=28081
IIOP_SSL_LISTENER_PORT=23821
ASADMIN_LISTENER_PORT=24849
IIOP_SSL_MUTUALAUTH_PORT=23921
JMX_SYSTEM_CONNECTOR_PORT=28687
HTTP_SSL_LISTENER_PORT=28182
IIOP_LISTENER_PORT=23701
Command create-local-instance executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-instance,
- create-node-config,
- create-node-ssh,
- create-system-properties,
- delete-local-instance,
- delete-system-property,
- list-instances,
- start-local-instance,
- stop-local-instance

Create-Managed-Executor-Service

Creates a managed executor service resource.

Synopsis

```
asadmin [asadmin-options] create-managed-executor-service [--help]
        [--enabled={false|true}]
        [--contextinfoenabled={false|true}]
        [--contextinfo={Classloader|JNDI|Security|WorkArea}]
        [--threadpriority threadpriority]
        [--longrunningtasks={false|true}]
        [--useforkjoinpool={false|true}]
        [--hungafterseconds hungafterseconds]
        [--corepoolsize corepoolsize]
        [--maximumpoolsize maximumpoolsize]
        [--keepaliveseconds keepaliveseconds]
        [--threadlifetimeseconds threadlifetimeseconds]
        [--taskqueuecapacity taskqueuecapacity]
        [--usevirtualthreads={false|true}]
        [--description description]
        [--property property]
        [--target target]
        jndi_name
```

Description

The `create-managed-executor-service` subcommand creates a managed executor service resource.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--enabled`**: Determines whether the resource is enabled at runtime.

+ The default value is `true`. **`--contextinfoenabled`**: Determines whether container contexts are propagated to threads. If set to `true`, the contexts specified in the `--contextinfo` option are propagated. If set to `false`, no contexts are propagated and the `--contextinfo` option is ignored.

+ The default value is `true`. **`--contextinfo`**: Specifies individual container contexts to propagate to threads. Valid values are Classloader, JNDI, Security, and WorkArea. Values are specified in a comma-separated list and are case-insensitive.

+ All contexts are propagated by default. **`--threadpriority`**: Specifies the priority to assign to created threads. The default value is 5. **`--longrunningtasks`**: Specifies

whether the resource should be used for long-running tasks. The default value is `false`. If set to `true`, long-running tasks are not reported as stuck. **`--hungafterseconds`**: Specifies the number of seconds that a task can execute before it is considered unresponsive. The default value is 0, which means that tasks are never considered unresponsive. **`--corepoolsize`**: Specifies the number of threads to keep in a thread pool, even if they are idle. The default value is 0. When a new task is submitted and the number of running threads is less than `corepoolsize`, a new thread is created to handle the request. When the value for `corepoolsize` is 0 (the default), new threads are never created unless the task queue is full or the resource is using direct queuing. Direct queuing occurs when `taskqueuecapacity` is 0, or when `taskqueuecapacity` is 2147483647 and `corepoolsize` is 0. **`--maximumpoolsize`**: Specifies the maximum number of threads that a thread pool can contain. The default value is 2147483647, which means that the thread pool is essentially unbounded and can contain any number of threads. **`--keepaliveseconds`**: Specifies the number of seconds that threads can remain idle when the number of threads is greater than `corepoolsize`.

+ The default value is 60. **`--threadlifetimeseconds`**: Specifies the number of seconds that threads can remain in a thread pool before being purged, regardless of whether the number of threads is greater than `corepoolsize` or whether the threads are idle.

+ The default value is 0, which means that threads are never purged. **`--taskqueuecapacity`**: Specifies the number of submitted tasks that can be stored in the task queue awaiting execution. The default value is 2147483647, which means that the task queue is essentially unbounded and can store any number of submitted tasks. **`--usevirtualthreads`**: If enabled, virtual threads will be used instead of the default platform thread pool. They provide quick creation, minimal overhead, and fast thread switching, but are executed on limited "carrying threads". Use virtual threads for tasks which depend on external resources like databases, micro-services, etc. Do not use them for long calculation.

+ The default value is `false`. **`--description`**: Descriptive details about the resource. **`--property`**: Optional attribute name/value pairs for configuring the resource.

+ Payara Server does not define any additional properties for this resource. Moreover, this resource does not currently use any additional properties. **`--target`**: Specifies the target for which you are creating the resource. Valid targets are:

+ `server`;; Creates the resource for the default server instance. This is the default value. `domain`;; Creates the resource for the domain. cluster_name;; Creates the resource for every server instance in the specified cluster. instance_name;; Creates the resource for the specified server instance.

Operands

jndi_name: The JNDI name of this resource.

Examples

Example 1 Creating a Managed Executor Service Resource

This example creates a managed executor service resource named `concurrent/myExecutor`.

```
asadmin> create-managed-executor-service --description "Executor Service example"
--useforkjoinpool concurrent/myExecutor
Managed executor service concurrent/myExecutor created successfully.
Command create-managed-executor-service executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-managed-executor-service,
- list-managed-executor-services

Create-Managed-Scheduled-Executor-Service

Creates a managed scheduled executor service resource.

Synopsis

```
asadmin [asadmin-options] create-managed-scheduled-executor-service [--help]
        [--enabled={false|true}]
        [--contextinfoenabled={false|true}]
        [--contextinfo={Classloader|JNDI|Security|WorkArea}]
        [--threadpriority threadpriority]
        [--longrunningtasks={false|true}]
        [--hungafterseconds hungafterseconds]
        [--corepoolsize corepoolsize]
        [--keepaliveseconds keepaliveseconds]
        [--threadlifetimeseconds threadlifetimeseconds]
        [--usevirtualthreads={false|true}]
        [--description description]
        [--property property]
        [--target target]
        jndi_name
```

Description

The `create-managed-scheduled-executor-service` subcommand creates a managed scheduled executor service resource.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--enabled`**: Determines whether the resource is enabled at runtime.

+ The default value is `true`. **`--contextinfoenabled`**: Determines whether container contexts are propagated to threads. If set to `true`, the contexts specified in the `--contextinfo` option are propagated. If set to `false`, no contexts are propagated and the `--contextinfo` option is ignored.

+ The default value is `true`. **`--contextinfo`**: Specifies individual container contexts to propagate to threads. Valid values are Classloader, JNDI, Security, and WorkArea. Values are specified in a comma-separated list and are case-insensitive.

+ All contexts are propagated by default. **`--threadpriority`**: Specifies the priority to

assign to created threads. The default value is 5. **`--longrunningtasks`**: Specifies whether the resource should be used for long-running tasks. The default value is `false`. If set to `true`, long-running tasks are not reported as stuck. **`--hungafterseconds`**: Specifies the number of seconds that a task can execute before it is considered unresponsive. The default value is 0, which means that tasks are never considered unresponsive. **`--corepoolsize`**: Specifies the number of threads to keep in a thread pool, even if they are idle. The default value is 0, which means that a thread is created when the first task is scheduled. **`--keepaliveseconds`**: Specifies the number of seconds that threads can remain idle when the number of threads is greater than `corepoolsize`.

+ The default value is 60. **`--threadlifetimeseconds`**: Specifies the number of seconds that threads can remain in a thread pool before being purged, regardless of whether the number of threads is greater than `corepoolsize` or whether the threads are idle.

+ The default value is 0, which means that threads are never purged. **`--usevirtualthreads`**: If enabled, virtual threads will be used instead of the default platform thread pool. They provide quick creation, minimal overhead, and fast thread switching, but are executed on limited "carrying threads". Use virtual threads for tasks which depend on external resources like databases, micro-services, etc. Do not use them for long calculation.

+ The default value is `false`. **`--description`**: Descriptive details about the resource. **`--property`**: Optional attribute name/value pairs for configuring the resource. Payara Server does not define any additional properties for this resource. Moreover, this resource does not currently use any additional properties. **`--target`**: Specifies the target for which you are creating the resource. Valid targets are:

+ `server`;; Creates the resource for the default server instance. This is the default value. `domain`;; Creates the resource for the domain. cluster_name;; Creates the resource for every server instance in the specified cluster. instance_name;; Creates the resource for the specified server instance.

Operands

jndi_name: The JNDI name of this resource.

Examples

Example 1 Creating a Managed Scheduled Executor Service Resource

This example creates a managed scheduled executor service resource named `concurrent/myScheduledExecutor`.

```
asadmin> create-managed-scheduled-executor-service --description "Scheduled
Executor example" concurrent/myScheduledExecutor
Managed scheduled executor service concurrent/myScheduledExecutor created
successfully.
Command create-managed-scheduled-executor-service executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-managed-scheduled-executor-service,
- list-managed-scheduled-executor-services

Create-Managed-Thread-Factory

Creates a managed thread factory resource.

Synopsis

```
asadmin [asadmin-options] create-managed-thread-factory [--help]
        [--enabled={false|true}]
        [--contextinfoenabled={false|true}]
        [--contextinfo={Classloader|JNDI|Security|WorkArea}]
        [--threadpriority threadpriority]
        [--usevirtualthreads={false|true}]
        [--description description]
        [--property property]
        [--target target]
        jndi_name
```

Description

The `create-managed-thread-factory` subcommand creates a managed thread

factory resource.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--enabled`**: Determines whether the managed thread factory is enabled at runtime.

+ The default value is `true`. **`--contextinfoenabled`**: Determines whether container contexts are propagated to threads. If set to `true`, the contexts specified in the `--contextinfo` option are propagated. If set to `false`, no contexts are propagated and the `--contextinfo` option is ignored.

+ The default value is `true`. **`--contextinfo`**: Specifies individual container contexts to propagate to threads. Valid values are Classloader, JNDI, Security, and WorkArea. Values are specified in a comma-separated list and are case-insensitive.

+ All contexts are propagated by default. **`--threadpriority`**: Specifies the priority to assign to created threads.

+ The default value is 5. **`--usevirtualthreads`**: If enabled, virtual threads will be used instead of the default platform thread pool. They provide quick creation, minimal overhead, and fast thread switching, but are executed on limited "carrying threads". Use virtual threads for tasks which depend on external resources like databases, micro-services, etc. Do not use them for long calculation.

+ The default value is `false`. **`--description`**: Descriptive details about the resource. **`--property`**: Optional attribute name/value pairs for configuring the resource. Payara Server does not define any additional properties for this resource. Moreover, this resource does not currently use any additional properties. **`--target`**: Specifies the target for which you are creating the resource. Valid targets are:

+ `server`;; Creates the resource for the default server instance. This is the default

value. `domain`;; Creates the resource for the domain. cluster_name;; Creates the resource for every server instance in the specified cluster. instance_name;; Creates the resource for the specified server instance.

Operands

jndi_name: The JNDI name of this resource.

Examples

Example 1 Creating a Managed Thread Factory Resource

This example creates a managed thread factory resource named `concurrent/myThreadFactory`.

```
asadmin> create-managed-thread-factory --description "Thread Factory example"
--threadpriority 1 concurrent/myThreadFactory
Managed thread factory concurrent/myThreadFactory created successfully.
Command create-managed-thread-factory executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-managed-thread-factory,
- list-managed-thread-factories

Create-Message-Security-Provider

Enables administrators to create a message security provider, which specifies how SOAP messages will be secured.

Synopsis

```
asadmin [asadmin-options] create-message-security-provider [--help]
[--target target]
--classname provider_class
[--layer message_layer] [--providertype provider_type]
[--requestauthsource request_auth_source ]
[--requestauthrecipient request_auth_recipient ]
[--responseauthsource response_auth_source ]
```

```
[--responseauthrecipient response_auth_recipient ]
[--isdefaultprovider] [--property name=value[:name=value]*]
provider_name
```

Description

The `create-message-security-provider` subcommand enables the administrator to create a message security provider for the security service which specifies how SOAP messages will be secured.

This command is supported in remote mode only.

Options

If an option has a short option name, then the short option precedes the long option name. Short options have one dash whereas long options have two dashes.

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which you are creating the message security provider. The following values are valid:
`server`;; Creates the provider for the default server instance `server` and is the default value. `domain`;; Creates the provider for the domain. cluster_name;; Creates the provider for every server instance in the cluster. instance_name;; Creates the provider for a particular sever instance. **`--classname`**: Defines the Java implementation class of the provider. Client authentication providers must implement the `com.sun.enterprise.security.jauth.ClientAuthModule` interface. Server-side providers must implement the `com.sun.enterprise.security jauth.ServerAuthModule` interface. A provider may implement both interfaces, but it must implement the interface corresponding to its provider type. **`--layer`**: The message-layer entity used to define the value of the `auth-layer` attribute of `message-security-config` elements. The default is `HttpServlet`. Another option is `SOAP`. **`--providertype`**: Establishes whether the provider is to be used as client authentication provider, server authentication provider, or both. Valid options for this property include `client`, `server`, or `client-server`. **`--requestauthsource`**: The `auth-source` attribute defines a requirement for message-layer sender authentication (e.g. username password) or

content authentication (e.g. digital signature) to be applied to request messages. Possible values are `sender` or `content`. When this argument is not specified, source authentication of the request is not required. **`--requestauthrecipient`**: The `auth-recipient` attribute defines a requirement for message-layer authentication of the receiver of a message to its sender (e.g. by XML encryption). Possible values are `before-content` or `after-content`. The default value is `after-content`. **`--responseauthsource`**: The `auth-source` attribute defines a requirement for message-layer sender authentication (e.g. username password) or content authentication (e.g. digital signature) to be applied to response messages. Possible values are `sender` or `content`. When this option is not specified, source authentication of the response is not required. **`--responseauthrecipient`**: The `auth-recipient` attribute defines a requirement for message-layer authentication of the receiver of the response message to its sender (e.g. by XML encryption). Possible values are `before-content` or `after-content`. The default value is `after-content`. **`--isdefaultprovider`**: The `default-provider` attribute is used to designate the provider as the default provider (at the layer) of the type or types identified by the `providertype` argument. There is no default associated with this option. **`--property`**: Use this property to pass provider-specific property values to the provider when it is initialized. Properties passed in this way might include key aliases to be used by the provider to get keys from keystores, signing, canonicalization, encryption algorithms, etc.

The following properties may be set:

`security.config`;; Specifies the location of the message security configuration file. To point to a configuration file in the domain-dir`/config` directory, use the system property `${com.sun.aas.instanceRoot}/config/`, for example: `${com.sun.aas.instanceRoot}/config/wss-server-config-1.0.xml`. The default is domain-dir`/config/ wss-serverconfig-1.0.xml`. `debug`;; If `true`, enables dumping of server provider debug messages to the server log. The default is `false`. `dynamic.username. password`;; If `true`, signals the provider runtime to collect the username and password from the `CallbackHandler` for each request. If `false`, the username and password for `wsse:UsernameToken`(s) is collected once, during

module initialization. This property is only applicable for a `ClientAuthModule`. The default is `false`. `encryption.key.alias`;; Specifies the encryption key used by the provider. The key is identified by its keystore alias. The default value is `s1as`. `signature.key.alias`;; Specifies the signature key used by the provider. The key is identified by its keystore alias. The default value is `s1as`.

Operands

provider_name: The name of the provider used to reference the `provider-config` element.

Examples

Example 1 Creating a Message Security Provider

The following example shows how to create a message security provider for a client.

```
asadmin> create-message-security-provider
--classname com.sun.enterprise.security.jauth.ClientAuthModule
--providertype client mySecurityProvider
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- delete-message-security-provider,
- list-message-security-providers

Create-Module-Config

Adds the default configuration of a module to `domain.xml`.

Synopsis

```
asadmin [asadmin-options] create-module-config [--help]
[--dryrun={false|true}]
[--all={false|true}]
[--target target]
[service_name]
```

Description

The `create-module-config` subcommand adds the default configuration of a module to `domain.xml`.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--dryrun`**: Displays the default configuration of a module but does not add it to `domain.xml`. The default value is `false`. **`--all`**: Adds all default configurations of modules to `domain.xml` if they are not already in it. The default value is `false`. **`--target`**: Specifies the target to which the default configuration is being added.
Possible values are as follows:
`server`;; Adds the default configuration to the default server instance. This is the default value. `domain`;; Adds the default configuration to the default domain. cluster-name;; Adds the default configuration to every server instance in the specified cluster. instance-name;; Adds the default configuration to the specified instance.

Operands

service_name: The name of the module for which the default configuration is to be added.

Examples

Example 1 Adding a Default Configuration to domain.xml

This example adds the default configuration of the web container module to `domain1` in `server-config` (the default configuration). Use the `--dryrun` option to preview the configuration before it is added.

```
asadmin> create-module-config web-container
Command create-module-config executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-module-config,
- get-active-module-config

Create-Network-Listener

Adds a new network listener socket.

Synopsis

```
asadmin [asadmin-options] create-network-listener [--help]
[--address address]
--listenerport listener-port
[--threadpool thread-pool]
--protocol protocol
[--transport transport]
[--enabled={true|false}]
[--jkenabled={false|true}]
[--target target]
listener-name
```

Description

The `create-network-listener` subcommand creates a network listener. This subcommand is supported in remote mode only.

NOTE	If you edit the special network listener named `admin-listener`, you must restart the server for the changes to take effect. The Administration Console does not tell you that a restart is required in this case.
NOTE	You can use the `create-http-listener` subcommand to create a network listener that uses the HTTP protocol without having to first create a protocol, transport, or HTTP configuration. This subcommand is a convenient shortcut, but it gives access to only a limited number of options.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--address`**: The IP address or the hostname (resolvable by DNS). **`--listenerport`**: The port number to create the listen socket on. Legal values are 1-65535. On UNIX, creating sockets that listen on ports 1-1024 requires superuser privileges. Configuring an SSL listen socket to listen on port 443 is standard. **`--threadpool`**: The name of the thread pool for this listener. Specifying a thread pool is optional. The default is `http-thread-pool`. **`--protocol`**: The name of the protocol for this listener. **`--transport`**: The name of the transport for this listener. Specifying a transport is optional. The default is `tcp`. **`--enabled`**: If set to `true`, the default, the listener is enabled at runtime. **`--jkenabled`**: If set to `true`, `mod_jk` is enabled for this listener. The default is false. **`--target`**: Creates the network listener only on the specified target. Valid values are as follows:
`server`;; Creates the network listener on the default server instance. This is the default value. configuration-name;; Creates the network listener in the specified configuration. cluster-name;; Creates the network listener on all server instances in the specified cluster. standalone-instance-name;; Creates the network listener on the specified standalone server instance.

Operands

listener-name: The name of the network listener.

Examples

Example 1 Creating a Network Listener

The following command creates a network listener named `sampleListener` that is not enabled at runtime:

```
asadmin> create-network-listener --listenerport 7272 --protocol http-1
--enabled=false sampleListener
Command create-network-listener executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-http-listener,
- set-network-listener-configuration,
- create-protocol,
- create-threadpool,
- create-transport,
- delete-network-listener,
- list-network-listeners

Create-Node-Config

Creates a node that is not enabled for remote communication.

Synopsis

```
asadmin [asadmin-options] create-node-config [--help]
[--nodehost node-host]
[--installdir as-install-parent] [--nodedir node-dir] node-name
```

Description

The `create-node-config` subcommand creates a node that is not enabled for remote communication. The `create-node-config` subcommand does not require the secure shell (SSH) to be configured to create the node.

A node represents a host on which the Payara Server software is installed. A node must exist for every host on which Payara Server instances reside.

NOTE	To represent the host where the DAS is running, Payara Server provides the predefined node `localhost-domain`. The predefined node `localhost-domain` is not enabled for remote communication.

All administration of instances on a node that is not enabled for remote communication must be performed on the host that the node represents. The domain administration server (DAS) on a remote host cannot contact the node. To administer instances on a node that represents a host that is remote from the DAS, you must use the following subcommands:

- create-local-instance
- delete-local-instance
- start-local-instance

However, you may use stop-local-instance or stop-instance to stop the instances.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--nodehost`**: The name of the host that the node represents. If this option is omitted, no host is specified for the node. **`--installdir`**: The full path to the parent of the base installation directory of the Payara Server software on the host, for example, `/export/payara5/`. If this option is omitted, no parent of the base installation directory of the Payara Server software is specified for the node. **`--nodedir`**: The path to the directory that is to contain Payara Server instances that are created on the node. If a relative path is specified, the path is relative to the as-install directory. If this option is omitted, no directory for instances is specified for the node.

Operands

node-name: The name of the node.
The name must meet the following requirements:
* The name may contain only ASCII characters. * The name must start with a letter, a number, or an underscore. * The name may contain only the following characters: **Lowercase letters** Uppercase letters **Numbers** Hyphen **Period** Underscore * The name must be unique in the domain and must not be the name of another node, a

cluster, a named configuration, or a Payara Server instance. * The name must not be `domain`, `server`, or any other keyword that is reserved by Payara Server.

Examples

Example 1 Creating a Node That Is Not Enabled for Remote Communication

This example creates the node `sj03` for host `sj03.example.com`. The node is not enabled for remote communication.

```
asadmin> create-node-config --nodehost sj03.example.com sj03

Command create-node-config executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-local-instance,
- create-node-ssh,
- delete-local-instance,
- delete-node-config,
- install-node,
- install-node-ssh,
- list-nodes,
- start-local-instance,
- stop-instance,
- stop-local-instance,
- uninstall-node,
- uninstall-node-ssh,

- update-node-config,
- update-node-ssh

Create-Node-Docker

Creates a node that hosts server instances on a Docker containers.

Synopsis

```
asadmin [asadmin-options] create-node-ssh [--help]
[--nodehost node-host]
[--installdir as-install-parent]
[--nodedir node-dir]
[--dockerpasswordfile filepath]
[--dockerimage imageName]
[--dockerport port]
[--useTls={false|true}]
node-name
```

Description

The `create-node-docker` subcommand creates a server node hosted on a Docker container.

A node represents a host on which the Payara Server software is installed. A node must exist for every host on which Payara Server instances reside.

The domain administration server (DAS) contacts a Docker node's host through the Docker engine's REST API on its specific configured host and port. By default, the subcommand fails and the node is not created if the DAS cannot reach the configured Docker engine's process.

The setup of a Docker node is more involved that other types of nodes, so it is recommended reading about the requirements and conditions for this command to work properly in creating-a-docker-node in Payara Server's High Availability section.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the

subcommand. `--nodehost`: The name of the host where the Docker engine process is listening through its REST API. This option is mandatory, otherwise the command fails. `--installdir`: The full path to the parent of the base installation directory of the Payara Server software on the host, for example, `/export/payara/`. The default is the parent of the default base installation directory of the Payara Server software for the DAS.

+ When creating a Docker node this parameter is mostly irrelevant when using the official Payara Server Node images. Use this parameter to point to custom server installations in case of using different images. `--nodedir`: The path to the directory that is to contain Payara Server instances that are created on the node. The default is `as-install/nodes`, where as-install is the base installation directory of the Payara Server software on the host.

+ If a relative path is specified, the path is relative to the `as-install` directory. `--dockerPasswordFile`: This is the fully-qualified path of the password file that the Docker instance will use for authentication against the DAS.

+ Please note, that this path should be the path to the file on the **remote** machine where . This file should be a standard Payara Server password file as you would use with asadmin. This must be specified, as Docker instances require secure admin to be enabled to start. `--dockerImage`: The Docker image to use for creating the container.

+ Defaults to `payara/server-node:7.2.0`. `--dockerPort`: The port where the Docker engine's REST API endpoint is exposed.

+ Defaults to `2376`. `--useTls`: Specifies whether the Docker engine's REST API endpoint is contacted through secure communication, as recommended in most environments. Defaults to `false`.

Operands

node-name: The name of the node.
The name must meet the following requirements:
* The name may contain only ASCII characters. * The name must start with a letter, a

number, or an underscore. * The name may contain only the following characters: **Lowercase letters** Uppercase letters **Numbers** Hyphen **Period** Underscore * The name must be unique in the domain and must not be the name of another node, a cluster, a named configuration, or a Payara Server instance. * The name must not be `domain`, `server`, or any other keyword that is reserved by Payara Server.

Examples

Example 1 Creating a Node

This example creates the node `docker-node-1` by contacting the Docker REST API which is hosted in `http://localhost:2376` (locally using the default port):

```
asadmin create-node-docker --nodehost localhost --dockerPasswordFile
/opt/passwordfile.txt --dockerport 2376 docker-node-1

Command create-node-ssh executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- delete-node-docker,
- list-nodes,
- update-node-docker

Create-Node-Ssh

Creates a node that is enabled for communication over SSH.

Synopsis

```
asadmin [asadmin-options] create-node-ssh [--help]
--nodehost node-host
[--installdir as-install-parent] [--nodedir node-dir]
[--sshport ssh-port] [--sshuser ssh-user]
[--sshkeyfile ssh-keyfile]
[--force={false|true}]
[--install={false|true}] [--archive archive]
```

```
node-name
```

Description

The `create-node-ssh` subcommand creates a node that is enabled for communication over secure shell (SSH).

A node represents a host on which the Payara Server software is installed. A node must exist for every host on which Payara Server instances reside.

The domain administration server (DAS) contacts an SSH node's host through the SSH connector to manage Payara Server instances that reside on the node. However, the DAS does not use the SSH connector to contact the host where the DAS is running because the DAS can run all `asadmin` subcommands locally.

By default, the subcommand fails and the node is not created if the DAS cannot contact the node's host through SSH. To force the node to be created in the DAS configuration even if the host cannot be contacted through SSH, set the `--force` option to `true`.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--nodehost`**: The name of the host that the node represents. The name of the host must be specified. Otherwise, an error occurs. **`--installdir`**: The full path to the parent of the base installation directory of the Payara Server software on the host, for example, `/export/payara/`. The default is the parent of the default base installation directory of the Payara Server software for the DAS. This default is useful only if Payara Server is installed in the same location on all hosts. **`--nodedir`**: The path to the directory that is to contain Payara Server instances that are created on the node. The default is `as-install/nodes`, where as-install is the base installation directory of the Payara Server software on the host. If a relative path is specified, the path is relative to the as-install directory. **`--sshport`**: The port to use for SSH connections to this node's host. The default is 22. If the `--nodehost` option is set to `localhost-`

`domain, the `--sshport` option is ignored. **`--sshuser`**: The user on this node's host that is to run the process for connecting to the host through SSH. The default is the user that is running the DAS process. To ensure that the DAS can read this user's SSH private key file, specify the user that is running the DAS process. If the `--nodehost` option is set to `localhost-domain`, the `--sshuser` option is ignored.
`--sshkeyfile`: The absolute path to the SSH private key file for user that the `--sshuser` option specifies. This file is used for authentication to the `sshd` daemon on the node's host.

> **NOTE** Payara Server also supports password authentication through the `AS_ADMIN_SSHPASSWORD` entry in the password file. The password file is specified in the `--paswordfile` option of the asadmin-1m utility.

If the SSH private key file is protected by a passphrase, the password file must contain the `AS_ADMIN_SSHKEYPASSPHRASE` entry.
The path to the key file must be reachable by the DAS and the key file must be readable by the DAS.
The default is the a key file in the user's `.ssh` directory. If multiple key files are found, the subcommand uses the following order of preference:

1. `id_rsa`
2. `id_dsa`
3. `identity`

`--force`: Specifies whether the node is created in the DAS configuration even if validation of the node's parameters fails. To validate a node's parameters, the DAS must be able to contact the node's host through SSH. Possible values are as follows:
`false`;; The node is not created if validation of the node's parameters fails (default).
`true`;; The node is created even if validation of the node's parameters fails. **`--install`**: Specifies whether the subcommand shall install the Payara Server software on the host that the node represents.
Possible values are as follows:

`false`;; The subcommand shall not install the Payara Server software on the host (default). `true`;; The subcommand shall install the Payara Server software on the host.
`--archive`: The absolute path to the archive file of the Payara Server software that is to be installed. If this option is omitted and the `--install` is `true`, the subcommand creates a ZIP archive of the Payara Server software from the installation where this subcommand is run. The archive does not contain the `domains` directory or the `nodes` directory.

Operands

node-name: The name of the node.
The name must meet the following requirements:
* The name may contain only ASCII characters. * The name must start with a letter, a number, or an underscore. * The name may contain only the following characters: **Lowercase letters** Uppercase letters **Numbers** Hyphen **Period** Underscore * The name must be unique in the domain and must not be the name of another node, a cluster, a named configuration, or a Payara Server instance. * The name must not be `domain`, `server`, or any other keyword that is reserved by Payara Server.

Examples

Example 1 Creating a Node

This example creates the node `adc` for the host `adc.example.com`. By default, the parent of the base installation directory of the Payara Server software is `/export/payara`.

```
asadmin> create-node-ssh
--nodehost adc.example.com
--installdir /export/payara adc

Command create-node-ssh executed successfully.
```

Example 2 Forcing the Creation of a Node

This example forces the creation of node `eg1` for the host `eghost.example.com`. The node is created despite the failure of the DAS to contact the host

`eghost.example.com` to validate the node's parameters.

```
asadmin> create-node-ssh --force --nodehost eghost.example.com eg1
Warning: some parameters appear to be invalid.
Could not connect to host eghost.example.com using SSH.
There was a problem while connecting to eghost.example.com:22
eghost.example.com
Continuing with node creation due to use of --force.

Command create-node-ssh executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-node-config,
- delete-node-ssh,
- install-node,
- install-node-ssh,
- list-nodes,
- ping-node-ssh,
- setup-ssh,
- uninstall-node,
- uninstall-node-ssh,
- update-node-ssh

Create-Password-Alias

Creates a password alias.

Synopsis

```
asadmin [asadmin-options] create-password-alias [--help]
aliasname
```

Description

The `create-password-alias` subcommand creates an alias for a password. An alias is a token of the form `${ALIAS=aliasname}`. The password that corresponds to the alias name is stored in an encrypted form.

The `create-password-alias` subcommand can be run interactively or noninteractively.

- When run interactively, the subcommand prompts the user for the alias password and to confirm the alias password.
- When run noninteractively, the subcommand reads the alias password from a file that is passed through the `--passwordfile` option of the asadmin-1m utility. The file must contain an entry of the form `{cprefix}ALIASPASSWORD`=alias-password, where alias-password is the alias password. The noninteractive form of this command is suitable for use in scripts.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

aliasname: Your choice of name for the password alias.

Examples

Example 1 Creating a Password Alias Interactively

This example creates the password alias `jmspassword-alias` interactively.

```
asadmin> create-password-alias jmspassword-alias
Enter the alias password>
Enter the alias password again>
Command create-password-alias executed successfully.
```

Example 2 Creating a Password Alias Noninteractively

This example uses the `--passwordfile` option of the `asadmin` utility to create the password alias `winuser` noninteractively.

```
$ asadmin --passwordfile aspwfile.txt create-password-alias winuser
Command create-password-alias executed successfully.
```

The file `aspwfile.txt` contains the following entry to specify the alias password:

```
AS_ADMIN_ALIASPASSWORD=sp@rky
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- delete-password-alias,
- list-password-aliases,
- update-password-alias

Create-Protocol-Filter

Adds a new protocol filter.

Synopsis

```
asadmin [asadmin-options] create-protocol-filter [--help]
--protocol protocol-name
--classname class-name
[--target server]
protocol-filter-name
```

Description

The `create-protocol-filter` subcommand creates a protocol filter for a protocol. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--protocol`**: The name of the associated protocol. **`--classname`**: The fully qualified name of the Java class that implements the protocol filter. **`--target`**: Creates the protocol filter only on the specified target. Valid values are as follows: `server`;; Creates the protocol filter on the default server instance. This is the default value. configuration-name;; Creates the protocol filter in the specified configuration. cluster-name;; Creates the protocol filter on all server instances in the specified cluster. standalone-instance-name;; Creates the protocol filter on the specified standalone server instance.

Operands

protocol-filter-name: The name of the protocol filter.

Examples

Example 1 Creating a Protocol Filter

The following command creates a protocol filter named `http1-filter`:

```
asadmin> create-protocol-filter --protocol http1
--classname com.company22.MyProtocolFilter http1-filter
Command create-protocol-filter executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-protocol,
- delete-protocol-filter,
- list-protocol-filters

Create-Protocol-Finder

Adds a new protocol finder.

Synopsis

```
asadmin [asadmin-options] create-protocol-finder [--help]
--protocol protocol-name
--targetprotocol target-protocol-name
--classname class-name
[--target server]
protocol-finder-name
```

Description

The `create-protocol-finder` subcommand creates a protocol finder for a protocol. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--protocol`**: The name of the associated protocol. **`--targetprotocol`**: The name of the target protocol. **`--classname`**: The fully qualified name of the Java class that implements the protocol finder. **`--target`**: Creates the protocol finder only on the specified target. Valid values are as follows: `server`;; Creates the protocol finder on the default server instance. This is the default value. configuration-name;; Creates the protocol finder in the specified configuration. cluster-name;; Creates the protocol finder on all server instances in the specified cluster. standalone-instance-name;; Creates the protocol finder on the specified standalone server instance.

Operands

protocol-finder-name: The name of the protocol finder.

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- delete-protocol-finder,
- list-protocol-finders

Create-Protocol

Adds a new protocol.

Synopsis

```
asadmin [asadmin-options] create-protocol [--help]
[--securityenabled={false|true}]
[--target target]
protocol-name
```

Description

The `create-protocol` subcommand creates a protocol. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--securityenabled`**: If set to `true`, the protocol runs SSL. You can turn SSL2 or SSL3 ON or OFF and set ciphers using an `ssl` element. The security setting globally enables or disables SSL by making certificates available to the server instance. The default value is `false`. **`--target`**: Creates the protocol only on the specified target. Valid values are as follows:
`server`;; Creates the protocol on the default server instance. This is the default value. configuration-name;; Creates the protocol in the specified configuration. cluster-name;; Creates the protocol on all server instances in the specified cluster. standalone-instance-name;; Creates the protocol on the specified standalone server instance.

Operands

protocol-name: The name of the protocol.

Examples

Example 1 Creating a Protocol

The following command creates a protocol named `http-1` with security enabled:

```
asadmin> create-protocol --securityenabled=true http-1
Command create-protocol executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- delete-protocol,
- list-protocols,
- create-network-listener

Create-Resource-Adapter-Config

Creates the configuration information for the connector module.

Synopsis

```
asadmin [asadmin-options] create-resource-adapter-config [--help]
[--threadpoolid threadpool]
[--objecttype object-type]
[--property (property-name=value)[:name=value]*]
raname
```

Description

The `create-resource-adapter-config` subcommand creates configuration information for the connector module. This subcommand can be run before deploying a resource adapter, so that the configuration information is available at the time of deployment. The resource adapter configuration can also be created after the resource adapter is deployed. In this case, the resource adapter is restarted with the new configuration. You must first create a thread pool, using the `create-threadpool` subcommand, and then identify that thread pool value as the ID in the `--threadpoolid` option.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--threadpoolid`**: The thread pool ID from which the work manager gets the thread. This option takes only one thread pool ID. **`--objecttype`**: The default is `user`. **`--property`**: Keyword-value pairs that specify additional configuration properties of the resource adapter Java bean. The keyword-value pairs are separated by a colon (`:`). The properties are the names of setter methods of the class that is referenced by the `resourceadapter-class` element in the `ra.xml` file.

Operands

raname: Indicates the connector module name. It is the value of the `resource-adapter-name` in the `domain.xml` file.

Examples

Example 1 Creating a Resource Adapter Con figuration

This example creates a resource adapter configuration for `ra1`.

```
asadmin> create-resource-adapter-config --property foo=bar --threadpoolid
mycustomerthreadpool ra1
Command create-resource-adapter-config executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-threadpool,
- delete-resource-adapter-config,
- list-resource-adapter-configs

Create-Resource-Ref

Creates a reference to a resource.

Synopsis

```
asadmin [asadmin-options] create-resource-ref [--help]
[--target target]
[--enabled={false|true}] reference_name
```

Description

The `create-resource-ref` subcommand creates a reference from a cluster or an unclustered server instance to a previously created resource, for example, a JDBC resource created by using the `create-jdbc-resource` subcommand. This effectively results in the resource being made available in the JNDI tree of the instance or cluster.

The target instance or instances making up the cluster need not be running or available for this subcommand to succeed. If one or more instances are not available, they will receive the new resource the next time they start.

NOTE

A `resource-ref` can only be created for bindable resources, such as a `jdbc-resource`, `connector-resource`, `admin-object-resource`, `mail-resource`, `custom-resource`, or `jndi-resource`.
A `jdbc-connection-pool` or a `connector-connection-pool` are not referred to directly by applications. Instead, they are referred to through a `jdbc-resource` or `connector-resource`, respectively.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which you are creating the resource reference. Valid targets are as follows:
`server`;; Creates the resource reference for the default server instance. This is the default target. cluster_name;; Creates the resource reference for every server instance in the cluster. instance_name;; Creates the resource reference for the named unclustered server instance. **`--enabled`**: Indicates whether the resource should be enabled. This value will take effect only if the resource is enabled at the global level. The default is

`true`.

Operands

reference_name: The name or JNDI name of the resource.

Examples

Example 1 Creating a Reference to a JMS Destination Resource

This example creates a reference to the JMS destination resource `jms/Topic` on the cluster `Cluster1`.

```
asadmin> create-resource-ref --target Cluster1 jms/Topic
resource-ref jms/Topic created successfully.
Command create-resource-ref executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- delete-resource-ref,
- list-resource-refs

Create-Service

configures the starting of a DAS or a Payara Server instance on an unattended boot

Synopsis

```
asadmin [asadmin-options] create-service [--help]
[--name service-name]
[--serviceproperties service-properties]
[--dry-run={false|true}] [--force={false|true}]
[--system-type system-type]
[--serviceuser service-user]
[--domaindir domain-dir]
[--nodedir node-dir] [--node node]
[domain-or-instance-name]
```

Description

The `create-service` subcommand configures the starting of a domain administration server (DAS) or a Payara Server instance on an unattended boot on Windows, Linux, and Oracle Solaris systems.

If no operand is specified and the domains directory contains only one domain, the subcommand configures the starting of the DAS for the default domain. If no operand is specified and the domains directory contains multiple domains, an error occurs.

If the operand specifies an instance, the `create-service` subcommand does not contact the domain administration server (DAS) to determine the node on which the instance resides.

To determine the node on which the instance resides, the subcommand searches the directory that contains the node directories. If multiple node directories exist, the node must be specified as an option of the subcommand.

The subcommand contains internal logic to determine whether the supplied operand is a DAS or an instance.

This subcommand is supported in local mode only.

Behavior of `create-service` on Windows Systems

On Windows systems, this subcommand must be run as the OS-level administrator user and requires the Microsoft .NET Framework (http://www.microsoft.com/net/), otherwise, the subcommand fails.

It creates a Windows service to represent the DAS or instance. The service is created in the disabled state. After this subcommand creates the service, you must use the Windows Services Manager or the Windows Services Wrapper to start, stop, uninstall, or install the service.

The subcommand creates the following Windows Services Wrapper files for the service in the `domain-dir\bin` directory or the `instance-dir\bin` directory:

- Configuration file: `service-nameService.xml`
- Executable file: `service-nameService.exe`

Behavior of `create-service` on Linux Systems

On Linux systems, the create-service subcommand first detects System-V or SystemD init systems.

On systems using System-V, it creates an initialization script `/etc/init.d/payara_domain-or-instance-name` and installs a link to this script in the `/etc/rc?.d` directories. After this subcommand creates the script, you must use this script to start, stop, or restart the domain or instance.

On systems using SystemD, the subcommand creates a configuration file `payara_domainname.service in /etc/systemd/system`. It is recommended to specify non-root user to run payara (parameter `--serviceuser`). It is possible to further configure the service via `--serviceproperties`.

On Linux systems, this subcommand must be run as the OS-level root user, because it writes to system directories.

Behavior of `create-service` on Oracle Solaris Systems

On Oracle Solaris systems, the `create-service` subcommand creates a Service Management Facility (SMF) service to represent the DAS or instance. The service is created in the disabled state. After this subcommand creates the service, you must use SMF commands to start, enable, disable, delete, or stop the service. For more information about SMF, see the following documentation for the Oracle Solaris operating system:

- Managing Services (Overview) in System Administration Guide: Basic Administration
- Managing Services (Tasks) in System Administration Guide: Basic Administration

On Oracle Solaris systems, this subcommand must be run as the OS-level user with superuser privileges. The configuration file for the DAS or instance must be stored in a

directory to which the superuser has access and cannot be stored on a network file system. The service that is created is controlled by the OS-level user who owns the directory where the configuration of the DAS or instance resides.

On Oracle Solaris systems, the manifest file is created in the following directory by default:

```
/var/svc/manifest/application/GlassFish/domain-or-instance-name_domain-or-instance-root-dir
```

To run this subcommand, you must have `solaris.smf.*` authorization. For information about how to grant authorizations to users, see the `useradd` and `usermod` man pages.

To run these commands as non-root user, the system administrator must be contacted so that the relevant authorizations are granted. You must also ensure that the following conditions are met:

- Oracle Solaris 10 administration commands such as `svccfg`, `svcs`, and `auths` are available through the `PATH` statement, so that these commands can be executed. A simple test to do so is to run the command `which svccfg` in the shell.
- You must have write permission for the path `/var/svc/manifest/application/GlassFish`. Usually, the superuser has write permission to this path.

If you delete a service that you created by using the `create-service` subcommand, you must delete the directory that contains the manifest file and the entire contents of the directory. Otherwise, an attempt to re-create the service by using the `create-service` subcommand fails.

The Oracle Solaris command `svccfg` does not delete this directory.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: `-?`: Displays the help text for the subcommand.

`--name`: (Windows and Oracle Solaris systems only) The name of the service that you will use when administering the service through Oracle Solaris SMF commands or the service management features of the Windows operating system. The default is the name of the domain or instance that is specified as the operand of this subcommand.

`--serviceproperties`: Specifies a colon(:)-separated list of various properties that are specific to the service.

+ To customize the display name of the service in the Windows Service list, set the `DISPLAY_NAME` property to the required name.

+ For Oracle Solaris 10 systems, if you specify `net_privaddr`, the service's processes will be able to bind to the privileged ports (<1024) on the platform. You can bind to ports< `1024` only if the owner of the service is superuser, otherwise, this is not allowed.

+ On Linux systems using SystemD these properties are stored as Environment entries, accessible via Microprofile property in the applications.

`--dry-run`: `-n`: Previews your attempt to create a service. Indicates issues and the outcome that will occur if you run the command without using the `--dry-run` option. Nothing is actually configured. Default is false.

`--force`: Specifies whether the service is created even if validation of the service fails.

+ Possible values are as follows:

+ `true`;; The service is created even if validation of the service fails. `false`;; The service is not created (default).

`--system-type`: If specified, system-type must be one of `systemd`, `systemv`, `solaris` or `windows`, which determine the type of init scripts to create.

+ If not specified, the command performs auto-detection of init system.

`--serviceuser`: (Linux systems only) The user that is to run the Payara Server software when the service is started. The default is the user that is running the subcommand. Specify this option if the Payara Server is to be run by a user other than the root user.

`--domaindir`: The absolute path of the directory on the disk that contains the configuration of the domain. If this option is specified, the operand must specify a domain.

`--nodedir`: Specifies the directory that contains the instance's node directory. The instance's files are stored in the instance's node directory.

+ The default is `as-install/nodes`. If this option is specified, the operand must specify an instance.

`--node`: Specifies the node on which the instance resides. This option may be omitted only if the directory that the `--nodedir` option specifies contains only one node directory. Otherwise, this option is required. If this option is specified, the operand must specify an instance.

Operands

domain-or-instance-name: The name of the domain or instance to configure. If no operand is specified, the default domain is used.

Examples

Example 1 Creating a Service on a Windows System

This example creates a service for the default domain on a system that is running Windows.

```
asadmin create-service
The Windows Service was created successfully.  It is ready to be started.  Here are
the details:
ID of the service: domain1
Display Name of the service:domain1 Payara Server
Server Directory: C:\payara7\glassfish\domains\domain1
Configuration file for Windows Services Wrapper: C:\payara7\glassfish\domains
\domain1\bin\domain1Service.xml
The service can be controlled using the Windows Services Manager or you can use the
```

```
Windows Services Wrapper instead:
Start Command:  C:\payara7\glassfish\domains\domain1\bin\domain1Service.exe  start
Stop Command:   C:\payara7\glassfish\domains\domain1\bin\domain1Service.exe  stop
Restart Command:  C:\payara7\glassfish\domains\domain1\bin\domain1Service.exe
restart
Uninstall Command:  C:\payara7\glassfish\domains\domain1\bin\domain1Service.exe
uninstall
Install Command:  C:\payara7\glassfish\domains\domain1\bin\domain1Service.exe
install
Status Command: C:\payara7\glassfish\domains\domain1\bin\domain1Service.exe status
You can also verify that the service is installed (or not) with sc query state= all
windows.services.uninstall.good=Found the Windows Service and successfully
uninstalled it.
For your convenience this message has also been saved to this file: C:\payara7
\glassfish\domains\domain1\PlatformServices.log
Command create-service executed successfully.
```

Example 2 Creating a Service on a Linux System using SystemD

This example creates a service for the default domain on a system that is running Linux and SystemD. It is necessary to inform SystemD about the new script by running systemctl daemon-reload. Then it is possible to start the service.

```
asadmin> create-service --service-user payaraadmin

The SystemD Service was created successfully. Here are the details:
Name of the service:domain1
Type of the service:Domain
Configuration location of the service:/etc/systemd/system/payara_domain1.service
User account that will run the service: payaraadmin
You have created the service but you need to start it yourself.  Here are the most
typical Linux commands of interest:

* systemctl daemon-reload
* systemctl start payara_domain1.service
* systemctl stop payara_domain1.service
* systemctl restart payara_domain1.service

For your convenience this message has also been saved to this file:
/opt/payara/glassfish/domains/domain1/PlatformServices.log

Command create-service executed successfully.
```

Example 3 Creating a Service on a Linux System using SystemV

This example creates a service for the default domain on a system that is running Linux and SystemV.

```
asadmin> create-service
```

```
Found the Linux Service and successfully uninstalled it.
The Service was created successfully. Here are the details:
Name of the service:domain1
Type of the service:Domain
Configuration location of the service:/etc/init.d/payara_domain1
User account that will run the service: root
You have created the service but you need to start it yourself.
Here are the most typical Linux commands of interest:

* /etc/init.d/payara_domain1 start
* /etc/init.d/payara_domain1 stop
* /etc/init.d/payara_domain1 restart

For your convenience this message has also been saved to this file:
/opt/payara/glassfish/domains/domain1/PlatformServices.log

Command create-service executed successfully.
```

Example 4 Creating a Service on an Oracle Solaris System

This example creates a service for the default domain on a system that is running Oracle Solaris.

```
asadmin> create-service

The Service was created successfully. Here are the details:
Name of the service:application/GlassFish/domain1
Type of the service:Domain
Configuration location of the service:/opt/payara/glassfish/domains
Manifest file location on the system:/var/svc/manifest/application
/GlassFish/domain1_opt_payara_glassfish_domains/Domain-service-smf.xml.
You have created the service but you need to start it yourself.
Here are the most typical Solaris commands of interest:
* /usr/bin/svcs -a | grep domain1 // status
* /usr/sbin/svcadm enable domain1 // start
* /usr/sbin/svcadm disable domain1 // stop
* /usr/sbin/svccfg delete domain1 // uninstall

Command create-service executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- _delete-service,
- `auths`,

- `svcs`
- `svccfg`,
- `useradd`,
- `usermod`
- "Managing Services (Overview)" in System Administration Guide: Basic Administration,
- "Managing Services (Tasks)" in System Administration Guide: Basic Administration Microsoft .NET Framework (`http://www.microsoft.com/net/`)

Create-Ssl

Creates and configures the SSL element in the selected HTTP listener, IIOP listener, or IIOP service.

Synopsis

```
asadmin [asadmin-options] create-ssl [--help]
[--target target]
--type listener_or_service_type
--certname cert_name
[--ssl2enabled={false|true}] [--ssl2ciphers ssl2ciphers]
[--ssl3enabled={true|false}] [--tlsenabled={true|false}]
[--ssl3tlsciphers ssl3tlsciphers]
[--tlsrollbackenabled={true|false}]
[--clientauthenabled={false|true}]
[listener_id]
```

Description

The `create-ssl` subcommand creates and configures the SSL element in the selected HTTP listener, IIOP listener, or IIOP service to enable secure communication on that listener/service.

This subcommand is supported in remote mode only.

Options

If an option has a short option name, then the short option precedes the long option name. Short options have one dash whereas long options have two dashes.

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target on which you are configuring the ssl element. The following values are valid:
`server`;; Specifies the server in which the iiop-service or HTTP/IIOP listener is to be configured for SSL. config;; Specifies the configuration that contains the HTTP/IIOP listener or iiop-service for which SSL is to be configured. cluster;; Specifies the cluster in which the HTTP/IIOP listener or iiop-service is to be configured for SSL. All the server instances in the cluster will get the SSL configuration for the respective listener or iiop-service. instance;; Specifies the instance in which the HTTP/IIOP listener or iiop-service is to be configured for SSL. **`--type`**: The type of service or listener for which the SSL is created. The type can be:
* `network-listener` * `http-listener` * `iiop-listener` * `iiop-service` * `jmx-connector`
When the type is `iiop-service`, the `ssl-client-config` along with the embedded `ssl` element is created in `domain.xml`. **`--certname`**: The nickname of the server certificate in the certificate database or the PKCS#11 token. The format of the name in the certificate is tokenname:nickname. For this property, the tokenname: is optional. **`--ssl2enabled`**: Set this property to `true` to enable SSL2. The default value is `false`. If both SSL2 and SSL3 are enabled for a virtual server, the server tries SSL3 encryption first. In the event SSL3 encryption fails, the server then tries SSL2 encryption. **`--ssl2ciphers`**: A comma-separated list of the SSL2 ciphers to be used. Ciphers not explicitly listed will be disabled for the target, even if those ciphers are available in the particular cipher suite you are using. If this option is not used, all supported ciphers are assumed to be enabled. Allowed values are:
* `rc4` * `rc4export` * `rc2` * `rc2export` * `idea` * `des` * `desede3` **`--ssl3enabled`**: Set this property to `false` to disable SSL3. The default value is `true`. If both SSL2 and SSL3 are enabled for a virtual server, the server tries SSL3 encryption first. In the event SSL3 encryption fails, the server then tries SSL2 encryption. **`--tlsenabled`**: Set this property to `false` to disable TLS. The default value is `true` It is good practice to enable TLS, which is a more secure version of SSL. **`--ssl3tlsciphers`**: A comma-separated

list of the SSL3 and/or TLS ciphers to be used. Ciphers not explicitly listed will be disabled for the target, even if those ciphers are available in the particular cipher suite you are using. If this option is not used, all supported ciphers are assumed to be enabled. Allowed values are:
* `SSL_RSA_WITH_RC4_128_MD5` * `SSL_RSA_WITH_3DES_EDE_CBC_SHA` * `SSL_RSA_WITH_DES_CBC_SHA` * `SSL_RSA_EXPORT_WITH_RC4_40_MD5` * `SSL_RSA_WITH_NULL_MD5` * `SSL_RSA_WITH_RC4_128_SHA` * `SSL_RSA_WITH_NULL_SHA` **`--tlsrollbackenabled`**: Set to `true` (default) to enable TLS rollback. TLS rollback should be enabled for Microsoft Internet Explorer 5.0 and 5.5. This option is only valid when `-tlsenabled`=`true`. **`--clientauthenabled`**: Set to `true` if you want SSL3 client authentication performed on every request independent of ACL-based access control. Default value is `false`.

Operands

listener_id: The ID of the HTTP or IIOP listener for which the SSL element is to be created. The listener_id is not required if the `--type` is `iiop-service`.

Examples

Example 1 Creating an SSL element for an HTTP listener

The following example shows how to create an SSL element for an HTTP listener named `http-listener-1`.

```
asadmin> create-ssl
--type http-listener
--certname sampleCert http-listener-1
Command create-ssl executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- delete-ssl

Create-System-Properties

Adds one or more system property elements that can be referenced elsewhere in the configuration.

Synopsis

```
asadmin [asadmin-options] create-system-properties [--help]
[--target target]
[name=value)[:name=value]*]
```

Description

The `create-system-properties` subcommand adds or updates system properties that can be referenced elsewhere on the server.

Payara Server provides hooks where tokens (system properties) can be specified. Because Payara Server does not have multiple server elements, you can specify a particular token at any level. When a domain supports multiple servers, the override potential can be exploited. When a domain is started or restarted, all `<system-property>` elements are resolved and available to the Java Virtual Machine by using the `System.setProperty()` call on each of them (with its name and value derived from the corresponding attributes of the element). This is analogous to sending the elements as `-D` parameters on the Java command line.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: The target on which you are creating the system properties.

Operands

target: The valid targets for this subcommand are instance, cluster, configuration, domain, and server. Server is the default option. Valid values are:
`server`;; Creates the properties on the default server instance. This is the default value.
`domain`;; Creates the properties for all server instances in the default domain.
configuration_name;; Creates the properties in the specified configuration.

cluster_name;; Creates the properties on all server instances in the specified cluster. instance_name;; Creates the properties on a specified server instance. **name=value**: The name value pairs of the system properties to add to the specified target. Multiple system properties must be separated by a : (colon). If a : (colon) appears in the name or value of a system property, it must be escaped with a \ (blackslash). If any system properties were previously defined, they are updated with the new values.

Examples

Example 1 Creating System Properties

This example creates a system property associated with an HTTP listener on a server instance named `myserver`.

```
asadmin> create-system-properties --target myserver http-listener-port=1088
Command create-system-properties executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-system-property,
- delete-system-property,
- list-system-properties

Create-System-Properties

Adds one or more system property elements that can be referenced elsewhere in the configuration.

Synopsis

```
asadmin [asadmin-options] create-system-property [--help]
[--target target]
[name=value]
```

Description

The `create-system-property` subcommand adds or updates a single system property that can be referenced elsewhere on the server.

Payara Server provides hooks where tokens (system properties) can be specified. Because Payara Server does not have multiple server elements, you can specify a particular token at any level. When a domain supports multiple servers, the override potential can be exploited. When a domain is started or restarted, all `<system-property>` elements are resolved and available to the Java Virtual Machine by using the `System.setProperty()` call on each of them (with its name and value derived from the corresponding attributes of the element). This is analogous to sending the elements as `-D` parameters on the Java command line.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: The target on which you are creating the system properties.

Operands

target: The valid targets for this subcommand are instance, cluster, configuration, domain, and server. Server is the default option. Valid values are:
`server`;; Creates the property on the default server instance. This is the default value.
`domain`;; Creates the property for all server instances in the default domain.
configuration_name;; Creates the property in the specified configuration. cluster_name;; Creates the property on all server instances in the specified cluster. instance_name;; Creates the property on a specified server instance. **name=value**: The name value pair of the system property to add to the specified target. If a system property was already defined, it is updated with the new value.

Examples

Example 1 Creating System Property

This example creates a system property associated with an HTTP listener on a server instance named `myserver`.

```
asadmin> create-system-property --target myserver http-listener-port=1088
Command create-system-property executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-system-properties,
- delete-system-property,
- list-system-properties

Create-Threadpool

Adds a thread pool.

Synopsis

```
asadmin [asadmin-options] create-threadpool [--help]
[--target target]
[--maxthreadpoolsize maxthreadpoolsize]
[--minthreadpoolsize minthreadpoolsize]
[--idletimeout idletimeout] [--maxqueuesize maxqueuesize]
threadpool-id
```

Description

The `create-threadpool` subcommand creates a thread pool with the specified name. You can specify maximum and minimum number of threads in the pool, the quantity of messages, and the idle timeout of a thread. The created thread pool can be used for servicing IIOP requests and for resource adapters to service work management requests. A thread pool can be used in multiple resource adapters.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target on which you are creating the thread pool.
Valid values are as follows:
`server`;; Creates the thread pool for the default Payara Server instance `server` and is the default value configuration-name;; Creates the thread pool for the named configuration. cluster-name;; Creates the thread pool for every instance in the cluster. instance-name;; Creates the thread pool for a particular instance.
`--maxthreadpoolsize`: Specifies the maximum number of threads the pool can contain. Default is 5. **`--minthreadpoolsize`**: Specifies the minimum number of threads in the pool. These are created when the thread pool is instantiated. Default is 2. **`--idletimeout`**: Specifies the amount of time in seconds after which idle threads are removed from the pool. Default is 900. **`--maxqueuesize`**: Specifies the maximum number of messages that can be queued until threads are available to process them for a network listener or IIOP listener. A value of -1 specifies no limit. Default is 4096.

Operands

threadpool-id: An ID for the work queue, for example, `threadpool-1`.

Examples

Example 1 Creating a Thread Pool

This command creates a new thread pool called `threadpool-1`.

```
asadmin> create-threadpool --maxthreadpoolsize 100
--minthreadpoolsize 20 --idletimeout 2 threadpool-1
Command create-threadpool executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- delete-threadpool,
- list-threadpools

Create-Transport

Adds a new transport.

Synopsis

```
asadmin [asadmin-options] create-transport [--help]
[--acceptorthreads acceptor-threads]
[--bytebuffertype byte-buffer-type]
[--classname class-name]
[--displayconfiguration={false|true}]
[--enablesnoop={false|true}]
[--idlekeytimeoutseconds idle-key-timeout]
[--maxconnectionscount max-connections]
[--readtimeoutmillis read-timeout]
[--writetimeoutmillis write-timeout]
[--selectorpolltimeoutmillis selector-poll-timeout]
[--tcpnodelay={false|true}]
[--target target]
transport-name
```

Description

The `create-transport` subcommand creates a transport for a network listener. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--acceptorthreads`**: The number of acceptor threads for the transport. The recommended value is the number of processors in the machine. The default value is `1`. **`--bytebuffertype`**: The type of the buffer to be provided for input streams created by a network-listener. Allowed values are `HEAP` and `DIRECT`. The default value is `HEAP`. **`--classname`**: The fully qualified name of the Java class that implements the transport. The default is `org.glassfish.grizzly.TCPSelectorHandler`. **`--displayconfiguration`**: If `true`, flushes the internal network configuration to the server log. Useful for debugging,

but reduces performance. The default is `false`. **`--enablesnoop`**: If `true`, writes request/response information to the server log. Useful for debugging, but reduces performance. The default is `false`. **`--idlekeytimeoutseconds`**: The idle key timeout. The default is `30` seconds. **`--maxconnectionscount`**: The maximum number of connections for the network listener that references this transport. A value of `-1` specifies no limit. The default value is `4096`. **`--readtimeoutmillis`**: The amount of time the server waits during the header and body parsing phase. The default is `30000` milliseconds, or 30 seconds. **`--writetimeoutmillis`**: The amount of time the server waits before considering the remote client disconnected when writing the response. The default is `30000` milliseconds, or 30 seconds. **`--selectorpolltimeoutmillis`**: The number of milliseconds a NIO Selector blocks waiting for events (user requests). The default value is `1000` milliseconds. **`--tcpnodelay`**: If `true`, the default, enables `TCP_NODELAY` (also called Nagle's algorithm). The default is `false`. **`--target`**: Creates the transport only on the specified target. Valid values are as follows:
`server`;; Creates the transport on the default server instance. This is the default value. configuration-name;; Creates the transport in the specified configuration. cluster-name;; Creates the transport on all server instances in the specified cluster. standalone-instance-name;; Creates the transport on the specified standalone server instance.

Operands

transport-name: The name of the transport.

Examples

Example 1 Creating a Transport

The following command creates a transport named `http1-trans` that uses a non-default number of acceptor threads:

```
asadmin> create-transport --acceptorthreads 100 http1-trans
Command create-transport executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-network-listener,
- delete-transport,
- list-transports

Create-Virtual-Server

Creates the named virtual server.

Synopsis

```
asadmin [asadmin-options] create-virtual-server [--help]
--hosts hosts
[--networklisteners network-listeners]
[--defaultwebmodule default-web-module]
[--state={on|off}]
[--logfile log-file]
[--property (name=value)[:name=value]*]
[--target target]
virtual-server-id
```

Description

The `create-virtual-server` subcommand creates the named virtual server. Virtualization in the Payara Server allows multiple URL domains to be served by a single HTTP server process that is listening on multiple host addresses. If the application is available at two virtual servers, they still share the same physical resource pools.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--hosts`**: A comma-separated (,) list of values allowed in the host request header to select the current virtual server. Each virtual server that is configured to the same connection group must have a unique host for that group. **`--networklisteners`**: A comma-separated (,) list of network listener IDs. Required only for a virtual server that is not the default virtual server. **`--defaultwebmodule`**:

The standalone web module associated with this virtual server by default. `--state`: Determines whether a virtual server is active (`on`) or inactive (`off` or disabled). Default is `on`. When inactive, the virtual server does not service requests. `--logfile`: Name of the file where log entries for this virtual server are to be written. By default, this is the server log. The file and directory in which the access log is kept must be writable by the user account under which the server runs. `--property`: Optional property name/value pairs for configuring the virtual server. The following properties are available: `sso-max-inactive-seconds`;; Specifies the number of seconds after which a user's single sign-on record becomes eligible for purging if no client activity is received. Since single sign-on applies across several applications on the same virtual server, access to any of the applications keeps the single sign-on record active. The default value is 300 seconds (5 minutes). Higher values provide longer single sign-on persistence for users, but at the expense of more memory use on the server. `sso-reap-interval-seconds`;; Specifies the number of seconds between purges of expired single sign-on records. The default value is 60. `setCacheControl`;; Specifies a comma-separated list of `Cache-Control` response directives. For a list of valid directives, see section 14.9 of the document at http://www.ietf.org/rfc/rfc2616.txt (`http://www.ietf.org/rfc/rfc2616.txt`). `allowLinking`;; If the value of this property is `true`, resources that are symbolic links will be served for all web applications deployed on this virtual server. Individual web applications may override this setting by using the property `allowLinking` under the `payara-web-app` element in the `payara-web.xml` file:

+

```
<payara-web-app>
<property name="allowLinking" value="[true|false]"/>
</payara-web-app>
```

IMPORTANT	sun-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

The default value is `true`.

`accessLogWriteInterval`

Indicates the number of seconds before the log will be written to the disk. The access log is written when the buffer is full or when the interval expires. If the value is 0 (zero), then the buffer is always written even if it is not full. This means that each time the server is accessed, the log message is stored directly to the file.

`accessLogBufferSize`

Specifies the size, in bytes, of the buffer where access log calls are stored.

`allowRemoteAddress`

This is a comma-separated list of regular expression patterns to which the remote client's IP address is compared. If this property is specified, the remote address must match for this request to be accepted. If this property is not specified, all requests will be accepted unless the remote address matches a `denyRemoteAddress` pattern. The default value for this property is null.

`denyRemoteAddress`

This is a comma-separated list of regular expression patterns to which the remote client's IP address is compared. If this property is specified, the remote address must not match for this request to be accepted. If this property is not specified, request acceptance is governed solely by the `allowRemoteAddress` property. The default value for this property is null.

`allowRemoteHost`

This is a comma-separated list of regular expression patterns to which the remote client's host name (as returned by `java.net.Socket.getInetAddress().getHostName()`) is compared. If this property is specified, the remote host name must match for this request to be accepted. If this property is not specified, all requests will be accepted unless the remote host name matches a denyRemoteHost pattern. The default value for this property is null.

`denyRemoteHost`

This is a comma-separated list of regular expression patterns to which the remote client's host name (as returned by `java.net.Socket.getInetAddress().getHostName()`) is compared. If this property is specified, the remote host name must not match for this request to be accepted. If this property is not specified, request acceptance is governed solely by the `allowRemoteHost` property. The default value for this property is null.

`authRealm`

Specifies the `name` attribute of an `auth-realm`, which overrides the server instance's default realm for standalone web applications deployed to this virtual server. A realm defined in a standalone web application's `web.xml` file overrides the virtual server's realm.

`securePagesWithPragma`

Set this property to `false` to ensure that for all web applications on this virtual server file downloads using SSL work properly in Internet Explorer.
You can set this property for a specific web application. For details, see payara-web-app in Payara Server Application Deployment section.

`contextXmlDefault`

Specifies the location, relative to domain-dir, of the `context.xml` file for this virtual server, if one is used. For more information about the `context.xml` file, see using-a-context.xml-file in the Payara Server Application Development section and The Context Container (`http://tomcat.apache.org/tomcat-5.5-doc/config/context.html`). Context parameters, environment entries, and resource definitions in `context.xml` are supported in the Payara Server.

`alternatedocroot_n`

Specifies an alternate document root (docroot), where n is a positive integer that allows specification of more than one. Alternate docroots allow web applications

to serve requests for certain resources from outside their own docroot, based on whether those requests match one (or more) of the URI patterns of the web application's alternate docroots.
If a request matches an alternate docroot's URI pattern, it is mapped to the alternate docroot by appending the request URI (minus the web application's context root) to the alternate docroot's physical location (directory). If a request matches multiple URI patterns, the alternate docroot is determined according to the following precedence order:

- Exact match
- Longest path match
- Extension match
  For example, the following properties specify three alternate docroots. The URI pattern of the first alternate docroot uses an exact match, whereas the URI patterns of the second and third alternate docroots use extension and longest path prefix matches, respectively.

```
<property name="alternatedocroot_1"
   value="from=/my.jpg dir=/srv/images/jpg"/>
<property name="alternatedocroot_2"
   value="from=*.jpg dir=/srv/images/jpg"/>
<property name="alternatedocroot_3"
   value="from=/jpg/* dir=/src/images"/>
```

The `value` of each alternate docroot has two components: The first component, `from`, specifies the alternate docroot's URI pattern, and the second component, `dir`, specifies the alternate docroot's physical location (directory). Spaces are allowed in the `dir` component.
You can set this property for a specific web application. For details, see payara-web-app in Payara Server Application Deployment section.

`send-error_n`

Specifies custom error page mappings for the virtual server, which are inherited by all web applications deployed on the virtual server. A web application can override

these custom error page mappings in its `web.xml` deployment descriptor. The value of each `send-error_n` property has three components, which may be specified in any order:

The first component, `code`, specifies the three-digit HTTP response status code for which the custom error page should be returned in the response.

The second component, `path`, specifies the absolute or relative file system path of the custom error page. A relative file system path is interpreted as relative to the `domain-dir/config` directory.

The third component, `reason`, is optional and specifies the text of the reason string (such as `Unauthorized` or `Forbidden`) to be returned.

For example:

```
<property name="send-error_1"
   value="code=401 path=/myhost/401.html reason=MY-401-REASON"/>
```

This example property definition causes the contents of `/myhost/401.html` to be returned with 401 responses, along with this response line:

```
HTTP/1.1 401 MY-401-REASON
```

`redirect_n`

Specifies that a request for an old URL is treated as a request for a new URL. These properties are inherited by all web applications deployed on the virtual server. The value of each `redirect_n` property has two components, which may be specified in any order:

The first component, `from`, specifies the prefix of the requested URI to match.

The second component, `url-prefix`, specifies the new URL prefix to return to the client. The from prefix is simply replaced by this URL prefix.

For example:

```
<property name="redirect_1"
value="from=/dummy url-prefix=http://etude"/>
```

`valve_n`

Specifies a fully qualified class name of a custom valve, where n is a positive integer that allows specification of more than one. The valve class must implement the `org.apache.catalina.Valve` interface from Tomcat or previous Payara Server releases, or the `org.glassfish.web.valve.GlassFishValve` interface from the current Payara Server release. For example:

```
<property name="valve_1"
   value="org.glassfish.extension.Valve"/>
```

You can set this property for a specific web application. For details, see payara-web-app in Payara Server Application Deployment section.

`listener_n`

Specifies a fully qualified class name of a custom Catalina listener, where n is a positive integer that allows specification of more than one. The listener class must implement the `org.apache.catalina.ContainerListener` or `org.apache.catalina.LifecycleListener` interface. For example:

```
<property name="listener_1"
   value="org.glassfish.extension.MyLifecycleListener"/>
```

You can set this property for a specific web application. For details, see payara-web-app in Payara Server Application Deployment section.

`errorReportValve`

Specifies a fully qualified class name of a custom valve that produces default error pages for applications on this virtual server. Specify an empty string to disable the default error page mechanism for this virtual server.

`cacheTTL`

Specifies the time to live (TTL), in milliseconds, for the virtual server cache used when serving files. Defaults to 5000. **`--target`**: Creates the virtual server only on

the specified target. Valid values are as follows:

`server`

Creates the virtual server on the default server instance. This is the default value.

configuration-name

Creates the virtual server in the specified configuration.

cluster-name

Creates the virtual server on all server instances in the specified cluster.

standalone-instance-name

Creates the virtual server on the specified standalone server instance.

Operands

virtual-server-id: Identifies the unique ID for the virtual server to be created. This ID cannot begin with a number.

Examples

Example 1 Creating a Virtual Server

The following command creates a virtual server named `sampleServer`:

```
asadmin> create-virtual-server --hosts pigeon,localhost
--property authRealm=ldap sampleServer
Command create-virtual-server executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- delete-virtual-server,
- list-virtual-servers,

- create-http-listener,
- create-network-listener,
- get,
- list,
- set

Debug-Asadmin

Variant of the `asadmin` utility for performing administrative tasks for Payara Server. This variant is useful for debugging local admin commands the launching of Payara Server. It suspends immediately waiting for a debug connection to port 9008.

Synopsis

```
debug-asadmin [--host host]
[--port port]
[--user admin-user]
[--passwordfile filename]
[--terse={true|false}]
[--secure={false|true}]
[--echo={true|false}]
[--interactive={true|false}]
[--detach={true|false}]
[--help]
[subcommand [options] [operands]]
```

Description

Use the `debug-asadmin` utility to debug local admin commands and the launching of the Payara Server. See the `asadmin` command for full documentation.

Delete-Admin-Object

Removes the administered object with the specified JNDI name.

Synopsis

```
asadmin [asadmin-options] delete-admin-object [--help]
[--target target] jndi_name
```

Description

The `delete-admin-object` subcommand removes an administered object with the

specified JNDI name.

This subcommand is supported in remote mote only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This is the name of the targets for which the administered object is to be deleted. Valid values are:

NOTE	Resources are always created for a domain as a whole but are only active for targets for which a `<resource-ref>` has been created using the `--target` option when the resource was created. This means that deleting a resource only deletes the `<resource-ref>` element for the specified `--target`, and does not delete the resource from the domain as a whole unless `domain` is specified as the `--target` for the deletion.

`server`

Deletes the administered object for the default server instance `server` and is the default value.

configuration_name

Deletes the administered object for the specified configuration.

cluster_name

Deletes the administered object for the specified cluster.

instance_name

Deletes the administered object for a particular server instance.

Operands

jndi_name: JNDI name of the administered object to be deleted.

Examples

Example 1 Deleting an Administered Object

This example deletes the administered object named `jms/samplelqueue`.

```
asadmin> delete-admin-object jms/samplequeue
Command delete-admin-object executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-admin-object,
- list-admin-objects

Delete-Application-Ref

Removes a reference to an application.

Synopsis

```
asadmin [asadmin-options] delete-application-ref [--help]
[--target target]
[--cascade=false] reference_name
```

Description

The `delete-application-ref` subcommand removes a reference from a cluster or an unclustered server instance to an application. This effectively results in the application element being undeployed and no longer available on the targeted instance or cluster.

The target instance or instances making up the cluster need not be running or available for this subcommand to succeed. If one or more instances are not available, they will no longer load the application the next time they start.

Removal of the reference does not result in removal of the application from the domain.

The bits are removed only by the `undeploy` subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target from which you are removing the application reference. Valid values are
* `server`- Specifies the default server instance as the target. `server` is the name of the default server instance and is the default value. * cluster_name - Specifies a certain cluster as the target. * instance_name - Specifies a certain stand-alone server instance as the target. **`--cascade`**: For a connector module, indicates whether the resources dependent on the module should also be recursively deleted. The default is `false`. The connector module can be either a stand-alone RAR file or a module within an EAR file.

Operands

reference_name: The name of the application or module, which can be a Java EE application module, Web module, EJB module, connector module or application client module.
The name can include an optional version identifier, which follows the name and is separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. To delete references to multiple versions, you can use an asterisk (`*`) as a wildcard character. For more information about module and application versions, see "module-and-application-versions" in Payara Server Application Deployment section.

Examples

Example 1 Deleting an Application Reference

The following example removes a reference to the Web module `MyWebApp` from the unclustered server instance `NewServer`.

```
asadmin> delete-application-ref --target NewServer MyWebApp
Command delete-application-ref executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-application-ref,
- list-application-refs,
- undeploy
- "Payara Server Application Deployment section"

Delete-Auth-Realm

Removes the named authentication realm.

Synopsis

```
asadmin [asadmin-options] delete-auth-realm [--help]
[--target target]
auth_realm-name
```

Description

The `delete-auth-realm` subcommand removes the named authentication realm. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target on which you are deleting the authentication realm. Valid values are `server`;; Deletes the realm for the default server instance `server` and is the default value. configuration_name;; Deletes the realm for the named configuration. cluster_name;; Deletes the realm for every server instance in the cluster.

instance_name;; Deletes the realm for a particular server instance.

Operands

auth_realm_name: Name of the realm to be deleted.

Examples

Example 1 Deleting an Authentication Realm

This example deletes the authentication realm `db`.

```
asadmin> delete-auth-realm db
Command delete-auth-realm executed successfully
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-auth-realm,
- list-auth-realms

Delete-Cluster

deletes a Payara Server cluster

Synopsis

```
asadmin [asadmin-options] delete-cluster [--help]
cluster-name
```

Description

The `delete-cluster` subcommand deletes a Payara Server cluster. A cluster can be deleted only if the cluster contains no Payara Server instances. If a cluster that you are deleting contains any instances, stop and delete the instances before deleting the cluster.

If the cluster's named configuration was created automatically for the cluster and no

other clusters or unclustered instances refer to the configuration, the configuration is deleted when the cluster is deleted.A configuration that is created automatically for a cluster is named cluster-name`-config`, where cluster-name is the name of the cluster.

This command is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

cluster-name: The name of the cluster to delete.

Examples

Example 1 Deleting a Payara Server Cluster

This example deletes the Payara Server cluster `adccluster`.

```
asadmin> delete-cluster adccluster
Command delete-cluster executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-cluster,
- delete-instance,
- delete-local-instance,
- list-clusters,
- start-cluster,

- stop-instance,
- stop-local-instance,
- stop-cluster,
- restart-cluster

Delete-Config-Property

Provides a way to delete a MicroProfile Configuration property in one of the built-in configuration sources.

Synopsis

```
asadmin [asadmin-options] delete-config-property [--help]
[--propertyName=<property.name>]
[--source={domain|config|server|application|module|cluster}]
[--sourceName=<source.name>]
[--moduleName=<module.name>]
[--target=<target[default:server]>]
```

Description

The `delete-config-property` deletes the record of a MicroProfile Configuration property in one of the built-in config sources.

The source is specified with `--source` and, where there is ambiguity, the `--sourceName` and `--moduleName` options can be used.

For example, where the source is `server`, the `--sourceName` can be used to specify the name of the server instance where the property is stored.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--propertyName`: The name of the configuration property to delete. **`--source`**: The configuration source where the property is stored. **`--sourceName`**: The name of the source when there may be ambiguity, for example a source of type `application` must specify the name of the application.

+ This property is required for sources of type: `config`, `server`, `application` or `module` **`--moduleName`**: The name of the module when the source is of type `module`.

+ When this is specified, the `sourceName` parameter must be provided and must have the name of the application where the module is deployed. **`--target`**: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Setting the configuration property value

This example deletes the `JMSBrokerURL` property from the configuration source named `myApplication`, belonging to the module `myModule`, and targeting the `myAppCluster` cluster.

```
asadmin delete-config-property --propertyName=JMSBrokerURL --source=module
--sourceName=myApplication --moduleName=myModule --target=myAppCluster
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-config-property
- get-config-property

Delete-Config

Deletes an existing named configuration.

Synopsis

```
asadmin [asadmin-options] delete-config [--help]
```

```
configuration-name
```

Description

The `delete-config` subcommand deletes an existing named configuration from the configuration of the domain administration server (DAS). You can delete a configuration only if no Payara Server instances or clusters refer to the configuration. A standalone configuration is automatically deleted when the instance or cluster that refers to it is deleted. You cannot delete the `default-config` configuration that is copied to create standalone configurations.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

configuration-name: The name of the configuration that you are deleting.

Examples

Example 1 Deleting a Named Configuration

This example deletes the named configuration `pmdconfig`.

```
asadmin> delete-config pmdconfig

Command delete-config executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- copy-config,

- list-configs
- configuration

Delete-Connector-Connection-Pool

Removes the specified connector connection pool.

Synopsis

```
asadmin [asadmin-options] delete-connector-connection-pool [--help]
[--cascade={false|true}] poolname
```

Description

The `delete-connector-connection-pool` subcommand removes the specified connector connection pool.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--cascade`**: When set to true, all connector resources associated with the pool, and the pool itself, are deleted. When set to false, the deletion of pool fails if any resources are associated with the pool. The resource must be deleted explicitly or the option must be set to true. Default is false.

Operands

poolname: The name of the connection pool to be removed.

Examples

Example 1 Deleting a Connector Connection Pool

This example deletes the connector connection pool named `jms/qConnPool`.

```
asadmin> delete-connector-connection-pool
--cascade=false jms/qConnPool
Command delete-connector-connection-pool executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-connector-connection-pool,
- list-connector-connection-pools,
- ping-connection-pool

Delete-Connector-Resource

Removes the connector resource with the specified JNDI name.

Synopsis

```
asadmin [asadmin-options] delete-connector-resource [--help]
[--target target] jndi_name
```

Description

The `delete-connector-resource` subcommand removes the connector resource with the specified JNDI name.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target from which you want to remove the connector resource. Valid targets are:

NOTE	Resources are always created for a domain as a whole but are only active for targets for which a `<resource-ref>` has been created using the `--target` option when the resource was created. This means that deleting a resource only deletes the `<resource-ref>` element for the specified `--target`, and does not delete the resource from the domain

> as a whole unless `domain` is specified as the `--target` for the deletion.

`server`

Deletes the connector resource from the default server instance. This is the default value.

`domain`

Deletes the connector resource from the domain.

cluster_name

Deletes the connector resource from every server instance in the cluster.

instance_name

Deletes the connector resource from a specified server instance.

Operands

jndi_name: The JNDI name of this connector resource.

Examples

Example 1 Deleting a Connector Resource

This example deletes a connector resource named `jms/qConnFactory`.

```
asadmin> delete-connector-resource jms/qConnFactory
Command delete-connector-resource executed successfully
```

Where `jms/qConnFactory` is the connector resource that is removed.

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-connector-resource,

- list-connector-resources

Delete-Connector-Security-Map

Celetes a security map for the specified connector connection pool.

Synopsis

```
asadmin [asadmin-options] delete-connector-security-map [--help]
--poolname connector_connection_pool_name mapname
```

Description

The `delete-connector-security-map` subcommand deletes a security map for the specified connector connection pool.

For this subcommand to succeed, you must have first created a connector connection pool using the `create-connector-connection-pool` subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--poolname`**: Specifies the name of the connector connection pool to which the security map that is to be deleted belongs.

Operands

mapname: Name of the security map to be deleted.

Examples

Example 1 Deleting a Connector Security Map

This example deletes `securityMap1` for the existing connection pool named `connector-pool1`.

```
asadmin> delete-connector-security-map
--poolname connector-pool1 securityMap1
Command delete-connector-security-map executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-connector-security-map,
- list-connector-security-maps,
- update-connector-security-map

Delete-Connector-Work-Security-Map

Deletes a work security map for the specified resource adapter.

Synopsis

```
asadmin [asadmin-options] delete-connector-work-security-map [--help]
--raname raname
mapname
```

Description

The `delete-connector-work-security-map` subcommand deletes a security map associated with the specified resource adapter. For this subcommand to succeed, you must have first created and deployed the specified resource adapter.

The enterprise information system (EIS) is any system that holds the data of an organization. It can be a mainframe, a messaging system, a database system, or an application.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--raname`**: Indicates the connector module name with which the work security map is associated.

Operands

mapname: The name of the work security map to be deleted.

Examples

Example 1 Deleting a Connector Work Security Map

This example deletes the work security map named `work_security_map_name` for the resource adapter named `ra_name`.

```
asadmin delete-connector-work-security-map
--raname ra_name work_security_map_name
Command delete-connector-work-security-map executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-connector-work-security-map,
- list-connector-work-security-maps,
- update-connector-work-security-map

Delete-Context-Service

Removes a context service resource with the specified JNDI name.

Synopsis

```
asadmin [asadmin-options] delete-context-service [--help]
[--target target]
context-service-name
```

Description

The `delete-context-service` subcommand removes a context service resource with the specified JNDI name.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--target`: Specifies the target from which you are deleting the resource. Valid targets are:

+ `server`;; Deletes the resource from the default server instance. This is the default value. `domain`;; Deletes the resource from the domain. cluster-name;; Deletes the resource from every server instance in the specified cluster. instance-name;; Deletes the resource from the specified server instance.

Operands

context-service-name: The JNDI name of the resource to be deleted.

Examples

Example 1 Deleting a Context Service Resource

This example deletes the context service resource named `concurrent/myContextService`.

```
asadmin> delete-context-service concurrent/myContextService
Context service concurrent/myContextService deleted successfully.
Command delete-context-service executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-context-service,
- list-context-services

Delete-Custom-Resource

Removes a custom resource.

Synopsis

```
asadmin [asadmin-options] delete-custom-resource [--help]
[--target target] jndi-name
```

Description

The `delete-custom-resource` subcommand removes a custom resource.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the location of the custom resources that you are deleting. Valid targets are server, domain, cluster, and instance. The default is server.

NOTE	Resources are always created for a domain as a whole but are only active for targets for which a `<resource-ref>` has been created using the `--target` option when the resource was created. This means that deleting a resource only deletes the `<resource-ref>` element for the specified `--target`, and does not delete the resource from the domain as a whole unless `domain` is specified as the `--target` for the deletion.

`server`

Deletes the resource for the default server instance. This is the default value.

`domain`

Deletes the resource for the domain.

cluster_name

Deletes the resource for every server instance in the cluster.

instance_name

Deletes the resource for a particular server instance.

Operands

jndi-name: The JNDI name of this resource.

Examples

Example 1 Deleting a Custom Resource

This example deletes a custom resource named `mycustomresource`.

```
asadmin> delete-custom-resource mycustomresource
Command delete-custom-resource executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-custom-resource,
- list-custom-resources

Delete-Deployment-Group

Deletes a Payara Server deployment group.

Synopsis

```
asadmin [asadmin-options] create-deployment-group [--help]
name
```

Description

The `delete-deployment-group` subcommand deletes a Payara Server deployment group. If the deployment group contains instances on it, they will not be deleted when the group is deleted.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: `-?`: Displays the help text for the subcommand.

Operands

name: The name of the deployment group to delete. If the name doesn't match any existing deployment groups the command will exit with an error.

Examples

Example 1 Deleting a Deployment Group

This example deletes a deployment group that is named `test-dgroup`:

```
asadmin delete-deployment-group test-dgroup
Command delete-deployment-group executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-deployment-group,
- create-instance,
- delete-instance,
- delete-local-instance,
- delete-application-ref,
- deploy,
- list-deployment-groups,
- start-deployment-group,

- stop-deployment-group,
- restart-deployment-group,
- add-instance-to-deployment-group
- remove-instance-from-deployment-group

Delete-Domain

Deletes a domain.

Synopsis

```
asadmin [asadmin-options] delete-domain [--help]
[--domaindir domaindir] domain-name
```

Description

The `delete-domain` subcommand deletes the specified domain. The domain must already exist and must be stopped.

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--domaindir`**: The directory where the domain to be deleted is located. If specified, the path must be accessible in the file system. If not specified, the domain under the domain root directory, which defaults to as-install`/domains`, is deleted.

Operands

domain-name: The unique name of the domain you want to delete.

Examples

Example 1 Deleting a Domain

This example deletes a domain named `mydomain4` from the default domains directory.

```
asadmin> delete-domain mydomain4
Domain mydomain4 deleted.
```

```
Command delete-domain executed successfully.
```

Example 2 deleting a Domain From an Alternate Location

This example deletes a domain named `sampleDomain` from the `/home/someuser/domains` directory.

```
asadmin> delete-domain --domaindir /home/someuser/domains sampleDomain
Domain sampleDomain deleted
Command delete-domain executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-domain,
- list-domains,
- start-domain,
- stop-domain

Delete-File-User

Removes the named file user.

Synopsis

```
asadmin [asadmin-options] delete-file-user [--help]
[--authrealmname auth_realm_name]
[--target target]
username
```

Description

The `delete-file-user` subcommand deletes the entry in the keyfile for the specified username.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--authrealmname`**: The name of the authentication realm with which the user was created. **`--target`**: This is the name of the target on which the command operates. The valid targets are:
`server`;; Deletes the file user on the default server instance. This is the default value
`domain`;; Deletes the file user in the domain. cluster_name;; Deletes the file user from every server instance in the cluster. instance_name;; Deletes the file user from a particular server instance.

Operands

username: This is the name of file user to be deleted.

Examples

Example 1 Deleting a User From a File Realm

The following example shows how to delete user named `sample_user` from a file realm.

```
asadmin> delete-file-user
sample_user
Command delete-file-user executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-file-user,
- list-file-groups,
- list-file-users,
- update-file-user

Delete-Http-Listener

Removes a network listener.

Synopsis

```
asadmin [asadmin-options] delete-http-listener [--help]
[--target target]
listener-id
```

Description

The `delete-http-listener` subcommand removes the specified network listener.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Deletes the network listener only from the specified target. Valid values are as follows:
`server`;; Deletes the network listener from the default server instance. This is the default value. configuration-name;; Deletes the network listener from the specified configuration. cluster-name;; Deletes the network listener from all server instances in the specified cluster. standalone-instance-name;; Deletes the network listener from the specified standalone server instance.

Operands

listener-id: The unique identifier for the network listener to be deleted.

Examples

Example 1 Using the delete-http-listener subcommand

The following command deletes the network listener named `sampleListener`:

```
asadmin> delete-http-listener sampleListener
Command delete-http-listener executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-http-listener
- list-http-listeners,
- restart-http-listeners

Delete-Http-Redirect

Removes an HTTP redirect.

Synopsis

```
delete-http-redirect [--help]
[--target target]
protocol-name
```

Description

The `delete-http-redirect` subcommand removes the specified HTTP redirect. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Deletes the HTTP redirect only from the specified target. Valid values are as follows:
`server`;; Deletes the HTTP redirect from the default server instance. This is the default value. configuration-name;; Deletes the HTTP redirect from the specified configuration. cluster-name;; Deletes the HTTP redirect from all server instances in the specified cluster. standalone-instance-name;; Deletes the HTTP redirect from the specified standalone server instance.

Operands

protocol-name: The name of the associated protocol.

Example 1 Deleting a HTTP Redirect

The following command deletes the HTTP redirect named `http-redirect`:

```
asadmin> delete-http-redirect http-redirect
Command delete-http-redirect executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-http-redirect

Delete-Http

Removes HTTP parameters from a protocol.

Synopsis

```
asadmin [asadmin-options] delete-http [--help]
[--target target]
protocol-name
```

Description

The `delete-http` subcommand removes the specified HTTP parameter set from a protocol. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Deletes the HTTP parameter set only from the specified target. Valid values are as follows:

`server`;; Deletes the HTTP parameter set from the default server instance. This is the

default value. configuration-name;; Deletes the HTTP parameter set from the specified configuration. cluster-name;; Deletes the HTTP parameter set from all server instances in the specified cluster. standalone-instance-name;; Deletes the HTTP parameter set from the specified standalone server instance.

Operands

protocol-name: The name of the protocol from which to delete the HTTP parameter set.

Examples

Example 1 Deleting an HTTP Parameter Set

The following command deletes the HTTP parameter set from a protocol named `http-1`:

```
asadmin> delete-http http-1
Command delete-http executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-http

Delete-Iiop-Listener

Removes an IIOP listener.

Synopsis

```
delete-iiop-listener [--help] [--target target] listener_id
```

Description

The `delete-iiop-listener` subcommand removes the specified IIOP listener. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target from which you are deleting the IIOP listener. Valid values are `server`;; Deletes the listener from the default server instance `server` and is the default value. configuration_name;; Deletes the listener from the named configuration. cluster_name;; Deletes the listener from every server instance in the cluster. instance_name;; Deletes the listener from a particular server instance.

Operands

listener_id: The unique identifier for the IIOP listener to be deleted.

Examples

Example 1 Deleting an IIOP Listener

The following command deletes the IIOP listener named `sample_iiop_listener`:

```
asadmin> delete-iiop-listener sample_iiop_listener
Command delete-iiop-listener executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-iiop-listener
- list-iiop-listeners

Delete-Instance

Deletes a Payara Server instance.

Synopsis

```
asadmin [asadmin-options] delete-instance [--help]
```

```
instance-name
```

Description

The `delete-instance` subcommand deletes a Payara Server instance. This subcommand requires the secure shell (SSH) to be configured on the host where the domain administration server (DAS) is running and on the host that is represented by the node where the instance resides.

NOTE

SSH is not required if the instance resides on a node of type `CONFIG` that represents the local host. A node of type `CONFIG` is not enabled for remote communication over SSH.

You may run this subcommand from any host that can contact the DAS.

The subcommand can delete any Payara Server instance, regardless of how the instance was created. For example, this subcommand can delete an instance that was created by using the create-local-instance subcommand.

The instance that is being deleted must not be running. Otherwise, an error occurs.

The subcommand deletes an instance by performing the following actions:

- Removing the instance from the configuration of the domain administration server (DAS)
- Deleting the instance's files from file system

If the instance that is being deleted is the only instance that is using the node directory, that directory is also removed.

If a standalone instance is deleted, the instance's standalone configuration is also deleted. A standalone instance refers to a configuration that is named instance-name`-config` to which no other clusters or unclustered instances refer.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

instance-name: The name of the instance to delete.

Examples

Example 1 Deleting a Payara Server Instance

This example deletes the Payara Server instance `pmdsainst`.

```
asadmin> delete-instance pmdsainst

Command delete-instance executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-instance
- create-local-instance
- delete-local-instance
- setup-ssh
- start-instance
- start-local-instance
- stop-instance
- stop-local-instance

Delete-Jacc-Provider

Enables administrators to delete JACC providers defined for a domain.

Synopsis

```
asadmin [asadmin-options] delete-jacc-provider [--help]
[--target target] jacc-provider-name
```

Description

The `delete-jacc-provider` subcommand enables administrators to delete JACC providers defined for a domain. JACC providers are defined as `jacc-provider` elements in the `security-service` element in the domain's `domain.xml` file. JACC providers can be created using the Payara Server Admin Console or the `create-jacc-provider` subcommand.

The default Payara Server installation includes two JACC providers, named `default` and `simple`. These default providers should not be deleted.

The JACC provider used byPayara Server for authorization is identified by the `jacc-provider` element of `security-service` in `domain.xml`. Therefore, if you delete the `jacc-provider` provider, make sure you change `jacc-provider` to the name of some other JACC provider that exists under `security-service`.

If you change the `jacc-provider` element to point to a different JACC provider, you must restart Payara Server.

This subcommand is supported in remote mode only.

Options

If an option has a short option name, then the short option precedes the long option name. Short options have one dash whereas long options have two dashes.

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target from which you are deleting the JACC provider. The following values are valid:

`server`;; Deletes the JACC provider on the default server instance. This is the default value. configuration_name;; Deletes the JACC provider in the specified configuration. cluster_name;; Deletes the JACC provider on all server instances in the specified cluster. instance_name;; Deletes the JACC provider on a specified server instance.

Operands

jacc-provider-name: The name of the JACC provider to be deleted.

Examples

Example 1 Deleting a JACC provider

The following example shows how to delete a JACC provider named `testJACC` from the default domain.

```
asadmin> delete-jacc-provider testJACC

Command delete-jacc-provider executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-jacc-provider,
- list-jacc-providers

Delete-Javamail-Resource

Removes a JavaMail session resource.

Synopsis

```
asadmin [asadmin-options] delete-javamail-resource [--help]
[--target target] jndi_name
```

Description

The `delete-javamail-resource` subcommand removes the specified JavaMail

session resource. Ensure that you remove all references to this resource before running this subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target from which you are deleting the JavaMail session resource. Valid values are:
`server`;; Deletes the resource from the default server instance. This is the default value. `domain`;; Deletes the resource from the domain. cluster_name;; Deletes the resource from every server instance in the cluster. instance_name;; Deletes the resource from a particular server instance.

Operands

jndi_name: The JNDI name of the JavaMail session resource to be deleted.

Examples

Example 1 Deleting a JavaMail Resource

This example deletes the JavaMail session resource named `mail/MyMailSession`.

```
asadmin> delete-javamail-resource mail/MyMailSession
Command delete-javamail-resource executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-javamail-resource,
- list-javamail-resources

Delete-Jdbc-Connection-Pool

Removes the specified JDBC connection pool.

Synopsis

```
asadmin [asadmin-options] delete-jdbc-connection-pool [--help]
[--cascade={false|true}]
jdbc_connection_pool_id
```

Description

The `delete-jdbc-connection-pool` subcommand deletes a JDBC connection pool. Before running this subcommand, all associations to the JDBC connection pool must be removed.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--cascade`**: If the option is set to true, all the JDBC resources associated with the pool, apart from the pool itself, are deleted. When set to false, the deletion of pool fails if any resources are associated with the pool. Resources must be deleted explicitly or the option must be set to true. The default value is false.

Operands

jdbc_connection_pool_id: The name of the JDBC resource to be removed.

Examples

Example 1 Deleting a JDBC Connection Pool

This example deletes the `sample_derby_pool` JDBC connection pool.

```
asadmin> delete-jdbc-connection-pool --cascade=false sample_derby_pool
Command delete-jdbc-connection-pool executed correctly.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-jdbc-connection-pool
- list-jdbc-connection-pools

Delete-Jdbc-Resource

Removes a JDBC resource with the specified JNDI name.

Synopsis

```
asadmin [asadmin-options] delete-jdbc-resource [--help]
[--target target] jndi_name
```

Description

The `delete-jdbc-resource` subcommand removes a JDBC resource. Ensure that all associations to the JDBC resource are removed before running this subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target from which you are removing the JDBC resource. Valid targets are:
`server`;; Removes the resource from the default server instance. This is the default value. `domain`;; Removes the resource from the domain. cluster_name;; Removes the resource from every server instance in the cluster. instance_name;; Removes the resource from a particular server instance.

NOTE	Resources are always created for a domain as a whole but are only active for targets for which a `<resource-ref>` has been created using the `--target` option when the resource was created. This means that deleting a resource only deletes the `<resource-ref>` element for the specified `--target`, and does not delete the resource from the domain

> as a whole unless `domain` is specified as the `--target` for the deletion.

Operands

jndi_name: The JNDI name of this JDBC resource to be removed.

Examples

Example 1 Deleting a JDBC Resource

The following example deletes the JDBC resource named `jdbc/DerbyPool`.

```
asadmin> delete-jdbc-resource jdbc/DerbyPool
Command delete-jdbc-resource executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-jdbc-resource,
- list-jdbc-resources

Delete-Jms-Host

Removes a JMS host.

Synopsis

```
asadmin [asadmin-options] delete-jms-host [--help]
[--target target]
jms_host_name
```

Description

The `delete-jms-host` subcommand removes the specified Java Message Service (JMS) host.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Deleting the default JMS host, named `default_JMS_host`, is not recommended.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Deletes the JMS host only from the specified target. Valid values are as follows:
`server`;; Deletes the JMS host from the default server instance. This is the default value. configuration-name;; Deletes the JMS host from the specified configuration. cluster-name;; Deletes the JMS host from every server instance in the specified cluster. instance-name;; Deletes the JMS host from the specified server instance.

Operands

jms_host_name: The name of the host to be deleted.

Examples

Example 1 Deleting a JMS host

The following subcommand deletes the JMS host named `MyNewHost`.

```
asadmin> delete-jms-host MyNewHost
Command delete-jms-host executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-jms-host,
- jms-ping,
- list-jms-hosts

Delete-Jms-Resource

Removes a JMS resource.

Synopsis

```
asadmin [asadmin-options] delete-jms-resource [--help]
[--target target]
jndi_name
```

Description

The `delete-jms-resource` subcommand removes the specified Java Message Service (JMS) resource. Ensure that you remove all references to this resource before executing this subcommand.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Deletes the JMS resource only from the specified target. Valid values are as follows:

> **NOTE**
>
> Resources are always created for a domain as a whole but are only active for targets for which a `<resource-ref>` has been created using the `--target` option when the resource was created. This means that deleting a resource only deletes the `<resource-ref>` element for the specified `--target`, and does not delete the resource from the domain as a whole unless `domain` is specified as the `--target` for the deletion.

`server`

Deletes the JMS resource from the default server instance. This is the default value.

`domain`

Deletes the JMS resource from the domain.

cluster-name

Deletes the JMS resource from every server instance in the specified cluster.

instance-name

Deletes the JMS resource from the specified server instance.

Operands

jndi_name: The JNDI name of the JMS resource to be deleted.

Examples

Example 1 Deleting a JMS destination resource

The following subcommand deletes the JMS destination resource named `jms/MyQueue`.

```
asadmin> delete-jms-resource jms/MyQueue
Administered object jms/MyQueue deleted.
Command delete-jms-resource executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-jms-resource,
- list-jms-resources

Delete-Jmsdest

Removes a JMS physical destination.

Synopsis

```
asadmin [asadmin-options] delete-jmsdest [--help]
--desttype type
[--target target]
dest_name
```

Description

The `delete-jmsdest` subcommand removes the specified Java Message Service (JMS) physical destination.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--desttype`**: The type of the JMS destination. Valid values are `topic` and `queue`. **`--target`**: Deletes the physical destination only from the specified target. Although the `delete-jmsdest` subcommand is related to resources, a physical destination is deleted using the JMS Service (JMS Broker), which is part of the configuration. A JMS Broker is configured in the config section of `domain.xml`. Valid values are as follows:
`server`;; Deletes the physical destination from the default server instance. This is the default value. configuration-name;; Deletes the physical destination from the specified configuration. cluster-name;; Deletes the physical destination from every server instance in the specified cluster. instance-name;; Creates the physical destination from the specified server instance.

Operands

dest_name: The unique identifier of the JMS destination to be deleted.

Examples

Example 1 Deleting a physical destination

The following subcommand deletes the queue named `PhysicalQueue`.

```
asadmin> delete-jmsdest --desttype queue PhysicalQueue
Command delete-jmsdest executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-jmsdest,
- flush-jmsdest,
- list-jmsdest

Delete-Jndi-Resource

Removes a JNDI resource.

Synopsis

```
asadmin [asadmin-options] delete-jndi-resource [--help]
[--target target] jndi_name
```

Description

The `delete-jndi-resource` subcommand removes the specified JNDI resource. You must remove all associations to the JNDI resource before running this subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Valid targets are described below.

NOTE

Resources are always created for a domain as a whole but are only active for targets for which a `<resource-ref>` has been created using the `--target` option when the resource was created. This means that deleting a resource only deletes the `<resource-ref>` element for the specified `--target`, and does not delete the resource from the domain as a whole unless `domain` is specified as the `--target` for the deletion.

`server`

Deletes the resource from the default server instance. This is the default value

`domain`

Deletes the resource from the domain

cluster_name

Deletes the resource for every server instance in the cluster

instance_name

Deletes the resource from the specified server instance

Operands

jndi_name: The name of the JNDI resource to be removed.

Examples

Example 1 Deleting a JNDI Resource

This example removes an existing JNDI resource named `sample_jndi_resource`.

```
asadmin> delete-jndi-resource sample_jndi_resource
Command delete-jndi-resource executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-jndi-resource,
- list-jndi-resources

Delete-Jvm-Options

Removes one or more options for the Java application launcher.

Synopsis

```
asadmin [asadmin-options] delete-jvm-options [--help]
[--target target]
(jvm-option-name[=jvm-option-value]) [:jvm-option-name[=jvm-option-name]]*
```

Description

The `delete-jvm-options` subcommand removes one or more command-line options for the Java application launcher. These options are removed from the Java configuration `java-config` element of the `domain.xml` file. To see the Java application launcher options that can be deleted, use the list-jvm-options subcommand.

The deletion of some options requires a server restart for changes to become effective. Other options are set immediately in the environment of the domain administration server (DAS) and do not require a restart.

Whether a restart is required depends on the type of option.

- Restart is not required for Java system properties whose names do not start with `-Djava.` or `-Djavax.` (including the trailing period). For example, restart is not required for the following Java system property:
  `-Denvironment=Production`
- Restart is required for the following options:
  - Java system properties whose names start with `-Djava.` or `-Djavax.` (including the trailing period). For example:
    `-Djava.security.manager`
  - Startup parameters for the Java application launcher. For example:
    `-client`
    `-Xmx1024m`
    `-d64`

To restart the DAS, use the restart-domain command.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target from which you are removing Java application launcher options.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Operands

jvm-option-name: One or more options delimited by a colon (:). The format of the operand depends on the following:
* If the option has a name and a value, the format is option-name=value. * If the option has only a name, the format is option-name. For example, `-Xmx2048m`.

NOTE	If an option name or option value contains a colon, the backslash (`\`) must be used to escape the colon in the name or value. Other characters might also require an escape character. For more information about escape characters in subcommand options, see the asadmin-1m man page.

Examples

Example 1 Deleting Java Application Launcher Options

This example removes multiple Java application launcher options.

```
asadmin> delete-jvm-options -Doption1=value1
"-Doption1=value1:-Doption2=value2"
Command delete-jvm-options executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-jvm-options,
- list-jvm-options,
- restart-domain

For more information about the Java application launcher, see the reference page for the operating system that you are using:

- Oracle Solaris and Linux: java - the Java application launcher (`http://java.sun.com/javase/7/docs/technotes/tools/solaris/java.html`)
- Windows: java - the Java application launcher (`http://java.sun.com/javase/7/docs/technotes/tools/windows/java.html`)

Delete-Local-Instance

Deletes a Payara Server instance on the machine where the subcommand is run/

Synopsis

```
asadmin [asadmin-options] delete-local-instance [--help]
[--nodedir node-dir] [--node node-name]
[instance-name]
```

Description

The `delete-local-instance` subcommand deletes a Payara Server instance on the machine where the subcommand is run. This subcommand does not require the secure shell (SSH) to be configured. You must run this command from the machine where the instance resides.

The subcommand can delete any Payara Server instance, regardless of how the instance was created. For example, this subcommand can delete an instance that was created by using the create-instance subcommand.

The instance that is being deleted must not be running. Otherwise, an error occurs.

The subcommand deletes an instance by performing the following actions:

- Removing the instance from the configuration of the domain administration server (DAS)
- Deleting the instance's files from file system

If the instance that is being deleted is the only instance that is using the node directory, that directory is also removed.

If a standalone instance is deleted, the instance's standalone configuration is also deleted. A standalone instance refers to a configuration that is named instance-name`-config` to which no other clusters or unclustered instances refer.

The `delete-local-instance` subcommand does not contact the DAS to determine the node on which the instance resides. To determine the node on which the instance resides, the subcommand searches the directory that contains the node directories. If multiple node directories exist, the node must be specified as an option of the subcommand.

If no operand is specified and only one instance resides on the specified node, the subcommand deletes the instance. If no operand is specified and multiple instances reside on the node, an error occurs.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--nodedir`**: Specifies the directory that contains the instance's node directory. The instance's files are stored in the instance's node directory. The default is as-install`/nodes`. **`--node`**: Specifies the node on which the instance resides. This option may be omitted only if the directory that the `--nodedir` option specifies contains only one node directory. Otherwise, this option is required.

Operands

instance-name: The name of the instance to delete. This operand may be omitted if only one instance resides on the specified node. Otherwise, this operand is required.

Examples

Example 1 Deleting an Instance

This example deletes the instance `pmdsainst`.

```
asadmin> delete-local-instance pmdsainst

Command delete-local-instance executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-instance,
- create-local-instance,
- delete-instance,
- start-instance,
- start-local-instance,
- stop-instance,
- stop-local-instance

Delete-Managed-Executor-Service

Removes a managed executor service resource with the specified JNDI name.

Synopsis

```
asadmin [asadmin-options] delete-managed-executor-service [--help]
[--target target]
managed_executor_service_name
```

Description

The `delete-managed-executor-service` subcommand removes a managed executor service resource with the specified JNDI name.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target from which you are deleting the resource. Valid targets are:

+ `server`;; Deletes the resource from the default server instance. This is the default value. `domain`;; Deletes the resource from the domain. cluster_name;; Deletes the resource from every server instance in the specified cluster. instance_name;; Deletes the resource from the specified server instance.

Operands

managed_executor_service_name: The JNDI name of the resource to be deleted.

Examples

Example 1 Deleting a Managed Executor Service Resource

This example deletes the managed executor service resource named `concurrent/myExecutor`.

```
asadmin> delete-managed-executor-service concurrent/Executor1
Managed executor service concurrent/Executor1 deleted successfully.
Command delete-managed-executor-service executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m

- create-managed-executor-service,
- list-managed-executor-services

Delete-Managed-Scheduled-Executor-Service

Removes a managed scheduled executor service resource with the specified JNDI name.

Synopsis

```
asadmin [asadmin-options] delete-managed-scheduled-executor-service [--help]
[--target target]
managed_scheduled_executor_service_name
```

Description

The `delete-managed-scheduled-executor-service` subcommand removes a managed scheduled executor service resource with the specified JNDI name.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target from which you are deleting the resource. Valid targets are:

+ `server`;; Deletes the resource from the default server instance. This is the default value. `domain`;; Deletes the resource from the domain. cluster_name;; Deletes the resource from every server instance in the specified cluster. instance_name;; Deletes the resource from the specified server instance.

Operands

managed_scheduled_executor_service_name: The JNDI name of the resource to be deleted.

Examples

Example 1 Deleting a Managed Scheduled Executor Service Resource

This example deletes the managed scheduled executor service resource named `concurrent/myScheduledExecutor`.

```
asadmin> delete-managed-scheduled-executor-service concurrent/ScheduledExecutor1
Managed scheduled executor service concurrent/ScheduledExecutor1 deleted
successfully.
Command delete-managed-scheduled-executor-service executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-managed-scheduled-executor-service,
- list-managed-scheduled-executor-services

Delete-Managed-Thread-Factory

Removes a managed thread factory resource with the specified JNDI name.

Synopsis

```
asadmin [asadmin-options] delete-managed-thread-factory [--help]
[--target target]
managed_thread_factory_name
```

Description

The `delete-managed-thread-factory` subcommand removes a managed thread factory resource with the specified JNDI name.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target from which you are deleting the resource. Valid targets are:

+ `server`;; Deletes the resource from the default server instance. This is the default value. `domain`;; Deletes the resource from the domain. cluster_name;; Deletes the resource from every server instance in the specified cluster. instance_name;; Deletes the resource from the specified server instance.

Operands

managed_thread_factory_name: The JNDI name of the resource to be deleted.

Examples

Example 1 Deleting a Managed Thread Factory Resource

This example deletes the managed thread factory resource named `concurrent/myThreadFactory`.

```
asadmin> delete-managed-thread-factory concurrent/Factory1
Managed thread factory concurrent/Factory1 deleted successfully.
Command delete-managed-thread-factory executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-managed-thread-factory,
- list-managed-thread-factories

Delete-Message-Security-Provider

Enables administrators to delete a message security provider.

Synopsis

```
asadmin [asadmin-options] delete-message-security-provider [--help]
[--target target]
--layer message_layer
provider_name
```

Description

The `delete-message-security-provider` subcommand enables administrators to delete a message security provider.

In terms of what happens when this subcommand is run, the `provider-config` sub-element for the given message layer (`message-security-config` element of `domain.xml` is deleted. The `domain.xml` file specifies parameters and properties to the Payara Server). The options specified in the list below apply to attributes within the `message-security-config` and `provider-config` sub-elements of the `domain.xml` file.

If the message-layer (`message-security-config` attribute) does not exist, it is created, and then the `provider-config` is created under it.

This command is supported in remote mode only.

Options

If an option has a short option name, then the short option precedes the long option name. Short options have one dash whereas long options have two dashes.

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target from which you are deleting the message security provider. Valid values are
`server`;; Deletes the message security provider from the default server instance `server` and is the default value `domain`;; Deletes the message security provider from the domain. cluster_name;; Deletes the message security provider from every server instance in the cluster. instance_name;; Deletes the message security provider from a particular sever instance. **`--layer`**: The message-layer from which the provider has to be deleted. The default value is `HttpServlet`.

Operands

provider_name: The name of the provider used to reference the `provider-config` element.

Examples

Example 1 Deleting a message security provider

The following example shows how to delete a message security provider for a client.

```
asadmin> delete-message-security-provider
--layer SOAP mySecurityProvider
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-message-security-provider,
- list-message-security-providers

Delete-Module-Config

Removes the configuration of a module from `domain.xml`.

Synopsis

```
asadmin [asadmin-options] delete-module-config [--help]
[--target target]
service_name
```

Description

The `delete-module-config` subcommand removes the configuration of a module from `domain.xml` and causes the module to use the default configuration included in the module.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the link:Technical Documentation/Payara Server Documentation/Command Reference/asadmin.adoc#asadmin-1m[`asadmin`] help page. **`--help`**: **`-?`**: Displays the

help text for the subcommand. `--target`: Specifies the target from which the configuration is to be deleted.
Possible values are as follows:
`server`;; Deletes the configuration from the default server instance. This is the default value. `domain`;; Deletes the configuration from the default domain. cluster-name;; Deletes the configuration from every server instance in the specified cluster. instance-name;; Deletes the configuration from the specified instance.

Operands

service_name: The name of the module for which configuration is to be removed.

Examples

Example 1 Deleting a Default Configuration From domain.xml

This example deletes the configuration of the web container module from `domain1` in `server-config` (the default configuration).

```
asadmin> delete-module-config web-container
Command delete-module-config executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-module-config,
- get-active-module-config

Delete-Network-Listener

Removes a network listener.

Synopsis

```
asadmin [asadmin-options] delete-network-listener [--help]
[--target target]
```

```
listener-name
```

Description

The `delete-network-listener` subcommand removes the specified network listener. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Deletes the network listener only from the specified target. Valid values are as follows:
`server`;; Deletes the network listener from the default server instance. This is the default value. configuration-name;; Deletes the network listener from the specified configuration. cluster-name;; Deletes the network listener from all server instances in the specified cluster. standalone-instance-name;; Deletes the network listener from the specified standalone server instance.

Operands

listener-name: The name of the network listener to be deleted.

Examples

Example 1 Deleting a Network Listener

The following command deletes the network listener named `sampleListener`:

```
asadmin> delete-network-listener sampleListener
Command delete-network-listener executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-network-listener,

- set-network-listener-configuration,
- list-network-listeners

Delete-Node-Config

Deletes a node that is not enabled for remote communication.

Synopsis

```
asadmin [asadmin-options] delete-node-config [--help]
node-name
```

Description

The `delete-node-config` subcommand deletes a node that is not enabled for remote communication from the domain. This subcommand does not require the secure shell (SSH) to be configured.

This subcommand can delete only a node that is not enabled for remote communication. The subcommand for deleting other types of nodes depends on the type of the node:

- A node that is enabled for remote communication over SSH must be deleted by using the delete-node-ssh subcommand.

To determine whether a node is enabled for remote communication, use the list-nodes subcommand.

No Payara Server instances must reside on the node that is being deleted. Otherwise, the subcommand fails. Before running this subcommand, delete any instances that reside on the node by using, for example, the delete-instance subcommand or the delete-local-instance subcommand.

NOTE | The predefined node `localhost-` pdomain cannot be deleted.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

node-name: The name of the node to delete. The node must not be enabled for communication over SSH. Otherwise, an error occurs.

Examples

Example 1 Deleting a Node That Is Not Enabled for Remote Communication

This example deletes the node `sj03`, which is not enabled for remote communication.

```
asadmin> delete-node-config sj03

Command delete-node-config executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-node-config,
- delete-instance,
- delete-local-instance,
- delete-node-ssh,
- install-node,
- install-node-ssh,
- list-nodes,
- uninstall-node,

- uninstall-node-ssh,
- update-node-config,
- update-node-ssh

Delete-Node-Docker

Deletes a Docker node

Synopsis

```
asadmin [asadmin-options] delete-node-docker [--help]
node-name
```

Description

The `delete-node-docker` subcommand deletes a Docker node. The command will assume that the node's configuration to be able to reach the container are still valid and will de-register the node configuration from the DAS and will automatically remove the container that corresponds to the deleted node.

If the settings of the Docker node are not valid or if the Docker engine is not reachable through its REST API endpoint, an error will occur and the command will fail.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

node-name: The name of the node to delete.

Examples

Example 1 Deleting a Docker Node

This example deletes the node `docker-node-1`:

```
asadmin> delete-node-docker docker-node-1
Command delete-node-docker executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-node-docker,
- delete-instance,
- delete-local-instance,
- list-nodes,
- update-node-docker

Delete-Node-Ssh

Deletes a node that is enabled for communication over SSH.

Synopsis

```
asadmin [asadmin-options] delete-node-ssh [--help]
[--uninstall={false|true}] [--force={false|true}]
node-name
```

Description

The `delete-node-ssh` subcommand deletes a node that is enabled for communication over secure shell (SSH) from the domain. This subcommand does not require SSH to be configured.

This subcommand can delete only a node that is enabled for communication over SSH. The subcommand for deleting other types of nodes depends on the type of the node:

- A node that is not enabled for remote communication must be deleted by using the delete-node-config subcommand.

To determine whether a node is enabled for communication over SSH, use the list-nodes

subcommand.

No Payara Server instances must reside on the node that is being deleted. Otherwise, the subcommand fails. Before running this subcommand, delete any instances that reside on the node by using, for example, the delete-instance subcommand or the delete-local-instance subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--uninstall`**: Specifies whether the Payara Server software is uninstalled from host that the node represents.
Possible values are as follows:
`false`;; The Payara Server software is not uninstalled from the host (default). `true`;; The Payara Server software is uninstalled from the host. By default, if any node except the predefined node `localhost-` domain resides on any host from which Payara Server software is being uninstalled, the subcommand fails. To uninstall the Payara Server software from a host on which user-defined nodes reside, set the `--force` option to `true`. If the `--force` option is `true`, the subcommand removes the entire content of the parent of the base installation directory. **`--force`**: If `--uninstall` is true, specifies whether the subcommand uninstalls the Payara Server software from a host even if a user-defined node resides on the host. Possible values are as follows:
`false`;; If a user-defined node resides on a host, the software is not uninstalled and the subcommand fails (default).
If the `--force` option is `false`, the subcommand removes only the Payara Server software files. Other content if the parent of the base installation directory, such as configuration files, are not removed. `true`;; The subcommand uninstalls the Payara Server software from the host even if a user-defined node resides on the host.
If the `--force` option is `true`, the subcommand removes the entire content of the parent of the base installation directory.

Operands

node-name: The name of the node to delete. The node must be enabled for communication over SSH. Otherwise, an error occurs.

Examples

Example 1 Deleting a Node That Is Enabled for Communication Over SSH

This example deletes the node `eg1`, which is enabled for communication over SSH.

```
asadmin> delete-node-ssh eg1
Command delete-node-ssh executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-node-ssh,
- delete-instance,
- delete-local-instance,
- delete-node-config,
- install-node,
- install-node-ssh,
- list-nodes,
- uninstall-node,
- uninstall-node-ssh,
- update-node-ssh

Delete-Password-Alias

Deletes a password alias.

Synopsis

```
asadmin [asadmin-options] delete-password-alias [--help]
aliasname
```

Description

This subcommand deletes a password alias.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

aliasname: This is the name of the substitute password as it appears in `domain.xml`.

Examples

Example 1 Deleting a Password Alias

```
asadmin>delete-password-alias
jmspassword-alias
```

Command delete-password-alias executed successfully

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-password-alias,
- list-password-aliases,
- update-password-alias

Delete-Protocol-Filter

Removes a protocol filter.

Synopsis

```
asadmin [asadmin-options] delete-protocol-filter [--help]
--protocol protocol-name
[--target server]
protocol-filter-name
```

Description

The `delete-protocol-filter` subcommand removes the specified protocol filter. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--protocol-name`**: The name of the associated protocol. **`--target`**: Deletes the protocol filter only from the specified target. Valid values are as follows: `server`;; Deletes the protocol filter from the default server instance. This is the default value. configuration-name;; Deletes the protocol filter from the specified configuration. cluster-name;; Deletes the protocol filter from all server instances in the specified cluster. standalone-instance-name;; Deletes the protocol filter from the specified standalone server instance.

Operands

protocol-filter-name: The name of the protocol filter to be deleted.

Examples

Example 1 Deleting a Protocol Filter

The following command deletes the protocol filter named `http1-filter`:

```
asadmin> delete-protocol-filter --protocol http1 http1-filter
Command delete-protocol-filter executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-protocol-filter,
- list-protocol-filters

Delete-Protocol-Finder

Removes a protocol finder.

Synopsis

```
asadmin [asadmin-options] delete-protocol-finder [--help]
--protocol protocol-name
[--target server]
protocol-finder-name
```

Description

The `delete-protocol-finder` subcommand removes the specified protocol finder. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--protocol-name`**: The name of the associated protocol. **`--target`**: Deletes the protocol finder only from the specified target. Valid values are as follows: `server`;; Deletes the protocol finder from the default server instance. This is the default value. configuration-name;; Deletes the protocol finder from the specified configuration. cluster-name;; Deletes the protocol finder from all server instances in the specified cluster. standalone-instance-name;; Deletes the protocol finder from the specified standalone server instance.

Operands

protocol-finder-name: The name of the protocol finder to be deleted.

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-protocol-finder,
- list-protocol-finders

Delete-Protocol

Removes a protocol.

Synopsis

```
asadmin [asadmin-options] delete-protocol [--help]
[--target target]
protocol-name
```

Description

The `delete-protocol` subcommand removes the specified protocol. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Deletes the protocol only from the specified target. Valid values are as follows:
`server`;; Deletes the protocol from the default server instance. This is the default value. configuration-name;; Deletes the protocol from the specified configuration. cluster-name;; Deletes the protocol from all server instances in the specified cluster. standalone-instance-name;; Deletes the protocol from the specified standalone server instance.

Operands

protocol-name: The name of the protocol to be deleted.

Examples

Example 1 Deleting a Protocol

The following command deletes the protocol named `http-1`:

```
asadmin> delete-protocol http-1
Command delete-protocol executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-protocol,
- list-protocols

Delete-Resource-Adapter-Config

Deletes the resource adapter configuration.

Synopsis

```
asadmin [asadmin-options] delete-resource-adapter-config [--help]
raname
```

Description

The `delete-resource-adapter-config` subcommand deletes the configuration information for the connector module.

This command is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand.

Operands

raname: Specifies the connector module name.

Examples

Example 1 Deleting a Resource Adapter Configuration

This example deletes the configuration information for `ra1`.

```
asadmin> delete-resource-adapter-config ra1
Command delete-resource-adapter-config executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-resource-adapter-config,
- list-resource-adapter-configs

Delete-Resource-Ref

Removes a reference to a resource.

Synopsis

```
asadmin [asadmin-options] delete-resource-ref [--help]
[--target target] reference_name
```

Description

The `delete-resource-ref` subcommand removes from a cluster or an unclustered server instance a reference to a resource (for example, a JDBC resource). This effectively results in the removal of the resource from the JNDI tree of the targeted instance or cluster.

The target instance or instances making up the cluster need not be running or available for this subcommand to succeed. If one or more instances are not available, they will no longer load the resource in the JNDI tree the next time they start.

Removal of the reference does not result in removal of the resource from the domain. The resource is removed only by the `delete` subcommand for that resource (for example, `delete-jdbc-resource`).

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target from which you are removing the resource reference. Valid values are `server`;; Removes the resource reference from the default server instance `server` and is the default value. cluster_name;; Removes the resource reference from every server instance in the cluster. instance_name;; Removes the resource reference from the named unclustered server instance.

Operands

reference_name: The name or JNDI name of the resource.

Examples

Example 1 Removing a Reference to a Resource

This example removes a reference to the JMS destination resource `jms/Topic` on the cluster `cluster1`.

```
asadmin> delete-resource-ref --target cluster1 jms/Topic
resource-ref jms/Topic deleted successfully.
Command delete-resource-ref executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-resource-ref,
- list-resource-refs

Delete-Service

Deletes the init scripts created by the `create-service` subcommand.

Synopsis

```
asadmin _delete-service [--help]
[--name service-name]
[--system-type system-type]
[--nodedir node-dir] [--node node]
[domain-or-instance-name]
```

Description

The _delete-service subcommand deletes init scripts created by the `create-script` subcommand.

If no operand is specified and the domains directory contains only one domain, the subcommand deletes the script starting the DAS for the default domain. If no operand is specified and the domains directory contains multiple domains, an error occurs.

The subcommand contains internal logic to smartly determine whether the supplied operand is a DAS or an instance.

This subcommand is supported in local mode only.

Options

`--help`: **`-?`**: Displays the help text for the subcommand.

`--name`: (Windows and Oracle Solaris systems only) The name of the service that you will use when administering the service through Oracle Solaris SMF commands or the service management features of the Windows operating system.

+ The default is the name of the domain or instance that is specified as the operand of this subcommand.

`--system-type`: If specified, system-type must be one of `systemd`, `systemv`, `solaris` or `windows`, which determine the type of init scripts to create.

+ If not specified, the command performs auto-detection of init system.

`--domaindir`: The absolute path of the directory on the disk that contains the configuration of the domain. If this option is specified, the operand must specify a domain.

`--nodedir`: Specifies the directory that contains the instance's node directory. The instance's files are stored in the instance's node directory. The default is `as-install/nodes`. If this option is specified, the operand must specify an instance.

`--node`: Specifies the node on which the instance resides. This option may be omitted only if the directory that the `--nodedir` option specifies contains only one node directory.

+ Otherwise, this option is required. If this option is specified, the operand must specify an instance.

Operands

domain-or-instance-name: The name of the domain or instance to configure. If no operand is specified, the default domain is used.

Examples

Example Deleting a Service on a Linux System using SystemD init

This example deletes a service for the default domain on a system that is running Linux and SystemD.

It is necessary to inform SystemD about deleting the script by running systemctl daemon-reload.

```
asadmin _delete-service
Command _delete-service executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-service

Delete-Ssl

Deletes the SSL element in the selected HTTP listener, IIOP listener, or IIOP service.

Synopsis

```
asadmin [asadmin-options] delete-ssl [--help]
[--target target]
--type listener_or_service_type
listener_id
```

Description

The `delete-ssl` subcommand deletes the SSL element in the selected HTTP listener, IIOP listener, or IIOP service.

The listener_id is not required if the `--type` is `iiop-service`.

This subcommand is supported in remote mode only.

Options

If an option has a short option name, then the short option precedes the long option name. Short options have one dash whereas long options have two dashes.

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target on which you are configuring the ssl element. The following values are valid:
`server`;; Specifies the server in which the iiop-service or HTTP/IIOP listener is to be unconfigured for SSL. config;; Specifies the configuration that contains the HTTP/IIOP listener or iiop-service for which SSL is to be unconfigured. cluster;; Specifies the cluster in which the HTTP/IIOP listener or iiop-service is to be unconfigured for SSL. All the server instances in the cluster will get SSL unconfigured for the respective listener or iiop-service. instance;; Specifies the instance in which the HTTP/IIOP listener or iiop-service is to be unconfigured for SSL. **`--type`**: The type of service or listener for which

the SSL is deleted. The type must be one of the following types:

- `http-listener`
- `iiop-listener`
- `iiop-service`

Operands

listener_id: The ID of the listener from which the SSL element is to be deleted.

The listener_id operand is not required if the `--type` is `iiop-service`.

Examples

Example 1 Deleting an SSL element from an HTTP listener

The following example shows how to delete an SSL element from an HTTP listener named `http-listener-1`.

```
asadmin> delete-ssl
--type http-listener http-listener-1
Command delete-ssl executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-ssl

Delete-System-Property

Removes a system property of the domain, configuration, cluster, or server instance, one at a time.

Synopsis

```
asadmin [asadmin-options] delete-system-property [--help]
[--target target_name ]
[property_name]
```

Description

The `delete-system-property` subcommand deletes a system property of a domain, configuration, cluster, or server instance. Make sure that the system property is not referenced elsewhere in the configuration before deleting it.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target on which you are deleting the system properties. The valid targets for this subcommand are instance, cluster, configuration, domain, and server. Server is the default option.

Operands

property_name: The name of the system property to remove.

Examples

Example 1 Deleting a System Property

This example deletes the system property named `http-listener-port`.

```
asadmin> delete-system-property http-listener-port
Command delete-system-property executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-system-properties,
- list-system-properties

Delete-Threadpool

Removes a thread pool.

Synopsis

```
asadmin [asadmin-options] delete-threadpool [--help]
[--target target] threadpool-id
```

Description

Removes the thread pool with the specified ID. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target from which you are removing the thread pool.
Valid values are as follows:
`server`;; Deletes the thread pool for the default Payara Server instance `server` and is the default value. configuration-name;; Deletes the thread pool for the named configuration. cluster-name;; Deletes the thread pool for every instance in the cluster. instance-name;; Deletes the thread pool for a particular instance.

Operands

threadpool-id: An ID for the work queue, for example, `thread-pool1`, `threadpool-2`, and so forth.

Examples

Example 1 Deleting a Thread Pool

This example deletes `threadpool-1`.

```
asadmin> delete-threadpool threadpool-1
Command delete-threadpool executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-threadpool,
- list-threadpools

Delete-Transport

Removes a transport.

Synopsis

```
asadmin [asadmin-options] delete-transport [--help]
[--target target]
transport-name
```

Description

The `delete-transport` subcommand removes the specified transport. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Deletes the transport only from the specified target. Valid values are as follows:
`server`;; Deletes the transport from the default server instance. This is the default value. configuration-name;; Deletes the transport from the specified configuration. cluster-name;; Deletes the transport from all server instances in the specified cluster. standalone-instance-name;; Deletes the transport from the specified standalone server instance.

Operands

transport-name: The name of the transport to be deleted.

Examples

Example 1 Deleting a Transport

The following command deletes the transport named `http1-trans`:

```
asadmin> delete-transport http1-trans
Command delete-transport executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-transport,
- list-transports

Delete-Virtual-Server

Removes a virtual server.

Synopsis

```
asadmin [asadmin-options] delete-virtual-server [--help]
[--target target] virtual-server-id
```

Description

The `delete-virtual-server` subcommand removes the virtual server with the specified virtual server ID. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Deletes the virtual server only from the specified target. Valid values are as follows:
`server`;; Deletes the virtual server from the default server instance. This is the default value. configuration-name;; Deletes the virtual server from the specified configuration. cluster-name;; Deletes the virtual server from all server instances in the specified cluster. standalone-instance-name;; Deletes the virtual server from the specified standalone server instance.

Operands

virtual-server-id: The unique identifier for the virtual server to be deleted.

Examples

Example 1 Deleting a Virtual Server

The following command deletes the virtual server named `sample_vs1`:

```
asadmin> delete-virtual-server sample_vs1
Command delete-virtual-server executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-virtual-server,
- list-virtual-servers

Deploy-Remote-Archive

Deploys the specified component from a remote location.

Synopsis

```
asadmin [asadmin-options] deploy-remote-archive [--help]
[--force={false|true}]
[--virtualservers virtual_servers]
[--contextroot context_root]
[--precompilejsp={false|true}]
[--verify={false|true}]
[--name component_name]
[--upload={true|false}]
[--retrieve local_dirpath]
[--dbvendorname dbvendorname]
[--deploymentplan deployment_plan]
[--altdd alternate_deploymentdescriptor]
[--runtimealtdd runtime_alternate_deploymentdescriptor]
[--deploymentorder deployment_order]
[--enabled={true|false}]
[--generatermistubs={false|true}]
[--availabilityenabled={false|true}]
[--asyncreplication={true|false}]
[--lbenabled={true|false}]
[--keepstate={false|true}]
```

```
[--libraries jar_file[,jar_file]*]
[--target target]
[--type pkg-type]
[--properties(name=value)[:name=value]*]
[--skipdsfailure]
[--additionalRepositories (repo1,repo2,...,repoN)]
[url|(groupId,artefactId,versionNumber)]
```

Description

The `deploy-remote-archive` subcommand deploys applications to the server and works in exactly the same as the "`deploy`" subcommand, but instead of accepting a local file path as its operand this command deploys an artifact that is available on a remote location via:

- A public URL where the artifact is hosted.
- A set of Maven GAV coordinates where the artifact is hosted.

This subcommand is supported in remote mode only.

WARNING	Although the command can accept a set Maven GAV coordinates, it does not actually make use of Maven, and so artefacts will not be saved into your local Maven repository.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--force`**: If set to `true`, redeploys the component even if the specified component has already been deployed or already exists. Default is `false`. **`--virtualservers`**: One or more virtual server IDs. Multiple IDs are separated by commas. **`--contextroot`**: Valid only if the archive is a web module. It is ignored for other archive types; it will be the value specified by default-context-path in web.xml, if specified; defaults to filename without extension. **`--precompilejsp`**: By default this option does not allow the JSP to be precompiled during deployment. Instead, JSPs are compiled during runtime. Default is `false`. **`--verify`**: If set to true and the required verifier packages are installed from the Update Tool, the syntax and semantics of the deployment descriptor is verified. Default is `false`. **`--name`**: Name of the deployable

component.

The name can include an optional version identifier, which follows the name and is separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. For more information about module and application versions, see "module-and-application-versions" in Payara Server Application Deployment section.

`--upload`: Specifies whether the subcommand uploads the file to the DAS. In most situations, this option can be omitted.

Valid values are as follows:

`false`;; The subcommand does not upload the file and attempts to access the file through the specified file name. If the DAS cannot access the file, the subcommand fails.

For example, the DAS might be running as a different user than the administration user and does not have read access to the file. In this situation, the subcommand fails if the `--upload` option is `false`. `true`;; The subcommand uploads the file to the DAS over the network connection.

The default value depends on whether the DAS is on the host where the subcommand is run or is on a remote host.

* If the DAS is on the host where the subcommand is run, the default is `false`. * If the DAS is on a remote host, the default is `true`.

If a directory filepath is specified, this option is ignored. **`--retrieve`**: Retrieves the client stub JAR file from the server machine to the local directory. **`--dbvendorname`**: Specifies the name of the database vendor for which tables are created. Supported values include `db2`, `mssql`, `h2`, `mysql`, `oracle`, `derby`, `javadb`, `postgresql`, and `sybase`. These values are case-insensitive. If not specified, the value of the `database-vendor-name` attribute in `glassfish-ejb-jar.xml` is used. If the connection cannot be established, or if the value is not valid, SQL-92 compliance is presumed.

`--deploymentplan`: Deploys the deployment plan, which is a JAR file that contains Payara Server descriptors. Specify this option when deploying a pure EAR file. A pure EAR file is an EAR without Payara Server descriptors. **`--altdd`**: Deploys the application using a Jakarta EE standard deployment descriptor that resides outside of the

application archive. Specify an absolute path or a relative path to the alternate deployment descriptor file. The alternate deployment descriptor overrides the top-level deployment descriptor packaged in the archive. For example, for an EAR, the `--altdd` option overrides `application.xml`. For a standalone module, the `--altdd` option overrides the top-level module descriptor such as `web.xml`. **`--runtimealtdd`**: Deploys the application using a Payara Server runtime deployment descriptor that resides outside of the application archive. Specify an absolute path or a relative path to the alternate deployment descriptor file. The alternate deployment descriptor overrides the top-level deployment descriptor packaged in the archive. For example, for an EAR, the `--runtimealtdd` option overrides `payara-application.xml`. For a standalone module, the `--runtimealtdd` option overrides the top-level module descriptor such as `glassfish-web.xml` (deprecated) or `payara-web.xml`. Applies to Payara Server deployment descriptors only (`glassfish-` **`.xml`** / `payara-` `.xml`); the name of the alternate deployment descriptor file must begin with `glassfish-` (or `payara-` in specific cases). Does not apply to `sun-` **`.xml`** `deployment descriptors, which are deprecated. *` `--deploymentorder`: Specifies the deployment order of the application. This is useful if the application has dependencies and must be loaded in a certain order at server startup. The deployment order is specified as an integer. The default value is 100. Applications with lower numbers are loaded before applications with higher numbers. For example, an application with a deployment order of 102 is loaded before an application with a deployment order of 110. If a deployment order is not specified, the default value of 100 is assigned. If two applications have the same deployment order, the first application to be deployed is the first application to be loaded at server startup.

The deployment order is typically specified when the application is first deployed but can also be specified or changed after initial deployment using the `set` subcommand. You can view the deployment order of an application using the `get` subcommand. **`--enabled`**: Allows users to access the application. If set to `false`, users will not be able to access the application. This option enables the application on the specified target instance or cluster. If you deploy to the target `domain`, this option is ignored, since deploying to the domain doesn't deploy to a specific instance or cluster. The

default is `true`. **`--generatermistubs`**: If set to `true`, static RMI-IIOP stubs are generated and put into the `client.jar`. If set to `false`, the stubs are not generated. Default is `false`. **`--availabilityenabled`**: This option controls whether high-availability is enabled for web sessions and for stateful session bean (SFSB) checkpointing and potentially passivation. If set to false (default) all web session saving and SFSB checkpointing is disabled for the specified application, web application, or EJB module. If set to true, the specified application or module is enabled for high-availability. Set this option to true only if high availability is configured and enabled at higher levels, such as the server and container levels. **`--asyncreplication`**: This option controls whether web session and SFSB states for which high availability is enabled are first buffered and then replicated using a separate asynchronous thread. If set to `true` (default), performance is improved but availability is reduced. If the instance where states are buffered but not yet replicated fails, the states are lost. If set to `false`, performance is reduced but availability is guaranteed. States are not buffered but immediately transmitted to other instances in the cluster. **`--lbenabled`**: This option controls whether the deployed application is available for load balancing. The default is true. **`--keepstate`**: This option controls whether web sessions, SFSB instances, and persistently created EJB timers are retained between re-deployments. The default is false. This option is supported only on the default server instance, named `server`. It is not supported and ignored for any other target.

Some changes to an application between re-deployments prevent this feature from working properly. For example, do not change the set of instance variables in the SFSB bean class.

For web applications, this feature is applicable only if in the `payara-web.xml` file the `persistence-type` attribute of the `session-manager` element is `file`.

For stateful session bean instances, the persistence type without high availability is set in the server (the `sfsb-persistence-type` attribute) and must be set to `file`, which is the default and recommended value.

If any active web session, SFSB instance, or EJB timer fails to be preserved or restored, none of these will be available when the redeployment is complete. However, the redeployment continues and a warning is logged.

To preserve active state data, Payara Server serializes the data and saves it in memory. To restore the data, the class loader of the newly redeployed application deserializes the data that was previously saved. **ifeval**: ["enterprise" == "community"] `--warlibs`: If set to true, enables an application to make use of shared libraries installed to `domain_dir/lib/warlibs`. Default is false. This parameter takes precedence over enabling warlibs in --properties. **endif**: [] `--libraries`: A comma-separated list of library JAR files. Specify the library JAR files by their relative or absolute paths. Specify relative paths relative to domain-dir`/lib/applibs`. The libraries are made available to the application in the order specified. `--target`: Specifies the target to which you are deploying. Valid values are:
`server`;; Deploys the component to the default server instance `server` and is the default value. `domain`;; Deploys the component to the domain. If `domain` is the target for an initial deployment, the application is deployed to the domain, but no server instances or clusters reference the application. If `domain` is the target for a redeployment (the `--force` option is set to true), and dynamic reconfiguration is enabled for the clusters or server instances that reference the application, the referencing clusters or server instances automatically get the new version of the application. If redeploying, and dynamic configuration is disabled, the referencing clusters or server instances do not get the new version of the application until the clustered or standalone server instances are restarted. cluster_name;; Deploys the component to every server instance in the cluster. instance_name;; Deploys the component to a particular stand-alone sever instance. `--type`: The packaging archive type of the component that is being deployed. Possible values are as follows:
`car`;; The component is packaged as a CAR file. `ear`;; The component is packaged as an EAR file. `ejb`;; The component is an EJB packaged as a JAR file. `osgi`;; The component is packaged as an OSGi bundle. `rar`;; The component is packaged as a RAR file. `war`;; The component is packaged as a WAR file. `--properties` **or** `--property`: Optional keyword-value pairs that specify additional properties for the deployment. The available properties are determined by the implementation of the component that is being deployed or redeployed. The `--properties` option and the `--property` option are equivalent. You can use either option regardless of the number of properties that

you specify.

You can specify the following properties for a deployment:

`jar-signing-alias`;; Specifies the alias for the security certificate with which the application client container JAR file is signed. Java Web Start will not run code that requires elevated permissions unless it resides in a JAR file signed with a certificate that the user's system trusts. For your convenience, Payara Server signs the JAR file automatically using the certificate with this alias from the domain's keystore. Java Web Start then asks the user whether to trust the code and displays the Payara Server certificate information. To sign this JAR file with a different certificate, add the certificate to the domain keystore, then use this property. For example, you can use a certificate from a trusted authority, which avoids the Java Web Start prompt, or from your own company, which users know they can trust. Default is `s1as`, the alias for the self-signed certificate created for every domain. `java-web-start-enabled`;; Specifies whether Java Web Start access is permitted for an application client module. Default is true. `compatibility`;; Specifies the Payara Server release with which to be backward compatible in terms of JAR visibility requirements for applications. The only allowed value is `v2`, which refers to Payara Enterprise Server version 2 or Sun Java System Application Server version 9.1 or 9.1.1. Beginning in Java EE 6, the Java EE platform specification imposed stricter requirements than Java EE 5 did on which JAR files can be visible to various modules within an EAR file. In particular, application clients must not have access to EJB JAR files or other JAR files in the EAR file unless references use the standard Java SE mechanisms (extensions, for example) or the Jakarta EE library-directory mechanism. Setting this property to `v2` removes these restrictions. `keepSessions={false|true}`;; Superseded by the `--keepstate` option.

If the `--force` option is set to `true`, this property can by used to specify whether active sessions of the application that is being redeployed are preserved and then restored when the redeployment is complete. Applies to HTTP sessions in a web container. Default is **`false`**.: **`false`**: Active sessions of the application are not preserved and restored (default). **`true`**: Active sessions of the application are preserved and restored.

If any active session of the application fails to be preserved or restored, none of the sessions will be available when the redeployment is complete. However, the redeployment continues and a warning is logged.
To preserve active sessions, Payara Server serializes the sessions and saves them in memory. To restore the sessions, the class loader of the newly redeployed application deserializes any sessions that were previously saved.
`preserveAppScopedResources`;; If set to `true`, preserves any application-scoped resources and restores them during redeployment. Default is `false`.

ifeval: ["enterprise" == "community"] `warlibs={false|true}`;; (This property will be overriden by `--warlibs`) If set to true, enables an application to make use of shared libraries installed to `domain_dir/lib/warlibs`. Default is `false`.
Libraries stored in this directory allow developers to create and deploy skinny WARs, as they are subject to additional annotation and extension scanning as well as processing for CDI, EJB, Servlet, and more. Unlike common and application libraries (those stored in `domain_dir/lib` and `domain_dir/lib/applibs` respectively), warlibs are cached and don't dynamically update if the files within the directory are modified. **endif**: []

```
  Other available properties are determined by the implementation of the component
that is being redeployed. +
  For components packaged as OSGi bundles (`--type=osgi`), the `deploy` subcommand
accepts properties arguments to wrap a WAR file as a WAB (Web Application Bundle)
at the time of deployment. The subcommand looks for a key named `UriScheme` and, if
present, uses the key as a URL stream handler to decorate the input stream. Other
properties are used in the decoration process. For example, the Payara Server OSGi
web container registers a URL stream handler named `webbundle`, which is used to
wrap a plain WAR file as a WAB. For more information about usage, see the example
in this help page.
*`--skipdsfailure`*:
This option will skip over a data source failure. This means an application can be
deployed with a data source set up, but the database need not be running at the
time.
```

+ Without this option the deployment would fail if a data source is defined and the database is not available at deployment time.

+

```
asadmin deploy --skipdsfailure /home/mark/clusterjsp-1.1.war
```

`--additionalRepositories`: This option is used to instruct the command additional Maven2 remote repositories on top of Maven Central that the command can use to look for the artifact when a set of GAV coordinates are used in the operand. The value this option takes is a comma-separated string of all remote repositories URLs, like this:

+

```
https://nexus.payara.fish/payara-artifacts,https://google.co.uk
```

+ NOTE: If an application artifact matching the GAV coordinates exists on multiple repositories, the first artifact found will be deployed. The order that the repositories are searched is the order that you specify them in.
For example, if this option's value is *repo1,repo2*, the order that the repositories would be searched is: *Maven Central* → *repo1* → *repo2*.

Operands

url|(groupId,artifactId,versionNumber): This operand supports 2 type of values:

+ * The URL where the archive that contains the application that is being deployed is located. This URL must correspond to publicly available HTTP(S) resources. * A set of Maven GAV coordinates (`groupId`, `artifactId`, `versionNumber`) where the application artifact can be located in a remote Maven repository. The three coordinates are specified in a comma-separated string.

+ TIP: By default, the command will look for a matching artifact in Maven Central. Additional Maven repositories can be configured using the `--additionalRepositories` option

Examples

Example 1 Deploying an Application hosted on a public HTTP(S) location

This example deploys the enterprise application packaged in the `clusterjsp-1.1.war` file to the default server instance `server`, which is hosted in the remote

location pointed by the `https://nexus.payara.fish/repository/payara-artifacts/fish/payara/testing/clusterjsp/1.1/clusterjsp-1.1.war` URL:

```
asadmin deploy-remote-archive --name clusterjsp
https://nexus.payara.fish/repository/payara-
artifacts/fish/payara/testing/clusterjsp/1.1/clusterjsp-1.1.war

Application deployed successfully with name clusterjsp.
Command deploy-remote-archive executed successfully
```

Example 2 Deploying a Web Application Using GAV coordinates

This example deploys the web application in the `clusterjsp-1.1.war` file to the default server instance `server`. This application can be located in the `fish.payara.testing:clusterjsp:1.1` GAV coordinates:

```
asadmin deploy-remote-archive --name clusterjsp fish.payara.testing,clusterjsp,1.1

Application deployed successfully with name clusterjsp.
Command deploy-remote-archive executed successfully
```

Example 3 Deploying a Web Application Using GAV coordinates using an additional remote repository

This example deploys the web application in the `clusterjsp-1.1.war` file to the default server instance `server` like the previous one, but configures the command to search for the Maven artifact on the remote repository `https://nexus.payara.fish/repository/payara-artifacts`:

```
asadmin deploy-remote-archive --name clusterjsp --additionalRepositories
https://nexus.payara.fish/repository/payara-artifacts
fish.payara.testing,clusterjsp,1.1

Application deployed successfully with name clusterjsp.
Command deploy-remote-archive executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get,
- deploy

Deploy

Deploys the specified component.

Synopsis

```
asadmin [asadmin-options] deploy [--help]
[--force={false|true}]
[--virtualservers virtual_servers]
[--contextroot context_root]
[--precompilejsp={false|true}]
[--verify={false|true}]
[--name component_name]
[--upload={true|false}]
[--retrieve local_dirpath]
[--dbvendorname dbvendorname]
[--deploymentplan deployment_plan]
[--altdd alternate_deploymentdescriptor]
[--runtimealtdd runtime_alternate_deploymentdescriptor]
[--deploymentorder deployment_order]
[--enabled={true|false}]
[--generatermistubs={false|true}]
[--availabilityenabled={false|true}]
[--asyncreplication={true|false}]
[--lbenabled={true|false}]
[--keepstate={false|true}]
[--libraries jar_file[,jar_file]*]
[--target target]
[--type pkg-type]
[--properties(name=value)[:name=value]*]
[--skipdsfailure]
[file_archive|filepath]
```

Description

The `deploy` subcommand deploys applications to the server. Applications can be enterprise applications, web applications, Enterprise JavaBeans (EJB) modules, connector modules, and application client modules. If the component is already deployed or already exists, it is forcibly redeployed if the `--force` option is set to `true` (default is `false`).

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--force`**: If set to `true`, redeploys the component even if the specified component has already been deployed or already exists. Default is `false`. **`--virtualservers`**: One or more virtual server IDs. Multiple IDs are separated by commas. **`--contextroot`**: Valid only if the archive is a web module. It is ignored for other archive types; it will be the value specified by default-context-path in web.xml, if specified; defaults to filename without extension. **`--precompilejsp`**: By default this option does not allow the JSP to be precompiled during deployment. Instead, JSPs are compiled during runtime. Default is `false`. **`--verify`**: If set to true and the required verifier packages are installed from the Update Tool, the syntax and semantics of the deployment descriptor is verified. Default is `false`. **`--name`**: Name of the deployable component.

The name can include an optional version identifier, which follows the name and is separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. For more information about module and application versions, see "module-and-application-versions" in Payara Server Application Deployment section. **`--upload`**: Specifies whether the subcommand uploads the file to the DAS. In most situations, this option can be omitted.

Valid values are as follows:

`false`;; The subcommand does not upload the file and attempts to access the file through the specified file name. If the DAS cannot access the file, the subcommand fails.

For example, the DAS might be running as a different user than the administration user and does not have read access to the file. In this situation, the subcommand fails if the `--upload` option is `false`. `true`;; The subcommand uploads the file to the DAS over the network connection.

The default value depends on whether the DAS is on the host where the subcommand is

run or is on a remote host.
* If the DAS is on the host where the subcommand is run, the default is `false`. * If the DAS is on the host where the subcommand is run, the default is `false`. * If the DAS is on a remote host, the default is `true`.
If a directory filepath is specified, this option is ignored. **`--retrieve`**: Retrieves the client stub JAR file from the server machine to the local directory. **`--dbvendorname`**: Specifies the name of the database vendor for which tables are created. Supported values include `db2`, `mssql`, `h2`, `mysql`, `oracle`, `derby`, `javadb`, `postgresql`, and `sybase`. These values are case-insensitive. If not specified, the value of the `database-vendor-name` attribute in `glassfish-ejb-jar.xml` is used. If the connection cannot be established, or if the value is not valid, SQL-92 compliance is presumed. **`--deploymentplan`**: Deploys the deployment plan, which is a JAR file that contains Payara Server descriptors. Specify this option when deploying a pure EAR file. A pure EAR file is an EAR without Payara Server descriptors. **`--altdd`**: Deploys the application using a Jakarta EE standard deployment descriptor that resides outside of the application archive. Specify an absolute path or a relative path to the alternate deployment descriptor file. The alternate deployment descriptor overrides the top-level deployment descriptor packaged in the archive. For example, for an EAR, the `--altdd` option overrides `application.xml`. For a standalone module, the `--altdd` option overrides the top-level module descriptor such as `web.xml`. **`--runtimealtdd`**: Deploys the application using a Payara Server runtime deployment descriptor that resides outside of the application archive. Specify an absolute path or a relative path to the alternate deployment descriptor file. The alternate deployment descriptor overrides the top-level deployment descriptor packaged in the archive. For example, for an EAR, the `--runtimealtdd` option overrides `payara-application.xml`. For a standalone module, the `--runtimealtdd` option overrides the top-level module descriptor such as `glassfish-web.xml` (deprecated) or `payara-web.xml`. Applies to Payara Server deployment descriptors only (`glassfish-` **`.xml`** / `payara-` `.xml`); the name of the alternate deployment descriptor file must begin with `glassfish-` (or `payara-` in specific cases). Does not apply to `sun-` **`.xml`** `deployment descriptors, which are deprecated. *` `--deploymentorder`: Specifies the deployment order of the application. This is useful if the application has dependencies and must be loaded in a

certain order at server startup. The deployment order is specified as an integer. The default value is 100. Applications with lower numbers are loaded before applications with higher numbers. For example, an application with a deployment order of 102 is loaded before an application with a deployment order of 110. If a deployment order is not specified, the default value of 100 is assigned. If two applications have the same deployment order, the first application to be deployed is the first application to be loaded at server startup.

The deployment order is typically specified when the application is first deployed but can also be specified or changed after initial deployment using the `set` subcommand. You can view the deployment order of an application using the `get` subcommand.

`--enabled`: Allows users to access the application. If set to `false`, users will not be able to access the application. This option enables the application on the specified target instance or cluster. If you deploy to the target `domain`, this option is ignored, since deploying to the domain doesn't deploy to a specific instance or cluster. The default is `true`. **`--generatermistubs`**: If set to `true`, static RMI-IIOP stubs are generated and put into the `client.jar`. If set to `false`, the stubs are not generated. Default is `false`. **`--availabilityenabled`**: This option controls whether high-availability is enabled for web sessions and for stateful session bean (SFSB) checkpointing and potentially passivation. If set to false (default) all web session saving and SFSB checkpointing is disabled for the specified application, web application, or EJB module. If set to true, the specified application or module is enabled for high-availability. Set this option to true only if high availability is configured and enabled at higher levels, such as the server and container levels. **`--asyncreplication`**: This option controls whether web session and SFSB states for which high availability is enabled are first buffered and then replicated using a separate asynchronous thread. If set to `true` (default), performance is improved but availability is reduced. If the instance where states are buffered but not yet replicated fails, the states are lost. If set to `false`, performance is reduced but availability is guaranteed. States are not buffered but immediately transmitted to other instances in the cluster. **`--lbenabled`**: This option controls whether the deployed application is available for load balancing. The default is true. **`--keepstate`**: This option controls whether web sessions, SFSB

instances, and persistently created EJB timers are retained between redeployments. The default is false. This option is supported only on the default server instance, named `server`. It is not supported and ignored for any other target.
Some changes to an application between redeployments prevent this feature from working properly. For example, do not change the set of instance variables in the SFSB bean class.
For web applications, this feature is applicable only if in the `payara-web.xml` file the `persistence-type` attribute of the `session-manager` element is `file`.
For stateful session bean instances, the persistence type without high availability is set in the server (the `sfsb-persistence-type` attribute) and must be set to `file`, which is the default and recommended value.
If any active web session, SFSB instance, or EJB timer fails to be preserved or restored, none of these will be available when the redeployment is complete. However, the redeployment continues and a warning is logged.
To preserve active state data, Payara Server serializes the data and saves it in memory. To restore the data, the class loader of the newly redeployed application deserializes the data that was previously saved. **ifeval**: ["enterprise" == "community"] **`--warlibs`**: If set to true, enables an application to make use of shared libraries installed to `domain_dir/lib/warlibs`. Default is false. This parameter takes precedence over enabling warlibs in --properties. **endif**: [] **`--libraries`**: A comma-separated list of library JAR files. Specify the library JAR files by their relative or absolute paths. Specify relative paths relative to domain-dir`/lib/applibs`. The libraries are made available to the application in the order specified. **`--target`**: Specifies the target to which you are deploying. Valid values are:
`server`;; Deploys the component to the default server instance `server` and is the default value. `domain`;; Deploys the component to the domain. If `domain` is the target for an initial deployment, the application is deployed to the domain, but no server instances or clusters reference the application. If `domain` is the target for a redeployment (the `--force` option is set to true), and dynamic reconfiguration is enabled for the clusters or server instances that reference the application, the referencing clusters or server instances automatically get the new version of the

application. If redeploying, and dynamic configuration is disabled, the referencing clusters or server instances do not get the new version of the application until the clustered or standalone server instances are restarted. cluster_name;; Deploys the component to every server instance in the cluster. instance_name;; Deploys the component to a particular stand-alone sever instance. **`--type`**: The packaging archive type of the component that is being deployed. Possible values are as follows: `car`;; The component is packaged as a CAR file. `ear`;; The component is packaged as an EAR file. `ejb`;; The component is an EJB packaged as a JAR file. `osgi`;; The component is packaged as an OSGi bundle. `rar`;; The component is packaged as a RAR file. `war`;; The component is packaged as a WAR file. **`--properties`** or **`--property`**: Optional keyword-value pairs that specify additional properties for the deployment. The available properties are determined by the implementation of the component that is being deployed or redeployed. The `--properties` option and the `--property` option are equivalent. You can use either option regardless of the number of properties that you specify.

You can specify the following properties for a deployment:

`jar-signing-alias`;; Specifies the alias for the security certificate with which the application client container JAR file is signed. Java Web Start will not run code that requires elevated permissions unless it resides in a JAR file signed with a certificate that the user's system trusts. For your convenience, Payara Server signs the JAR file automatically using the certificate with this alias from the domain's keystore. Java Web Start then asks the user whether to trust the code and displays the Payara Server certificate information. To sign this JAR file with a different certificate, add the certificate to the domain keystore, then use this property. For example, you can use a certificate from a trusted authority, which avoids the Java Web Start prompt, or from your own company, which users know they can trust. Default is `s1as`, the alias for the self-signed certificate created for every domain. `java-web-start-enabled`;; Specifies whether Java Web Start access is permitted for an application client module. Default is true. `compatibility`;; Specifies the Payara Server release with which to be backward compatible in terms of JAR visibility requirements for applications. The only allowed value is `v2`, which refers to Payara Enterprise Server version 2 or Sun Java

System Application Server version 9.1 or 9.1.1. Beginning in Java EE 6, the Java EE platform specification imposed stricter requirements than Java EE 5 did on which JAR files can be visible to various modules within an EAR file. In particular, application clients must not have access to EJB JAR files or other JAR files in the EAR file unless references use the standard Java SE mechanisms (extensions, for example) or the Jakarta EE library-directory mechanism. Setting this property to `v2` removes these restrictions. `keepSessions={false|true}`;; Superseded by the `--keepstate` option.
If the `--force` option is set to `true`, this property specifies whether active sessions of the application being redeployed are preserved and restored after redeployment. Applies to HTTP sessions in a web container. Default is `false`.
`false`: Active sessions of the application are not preserved and restored (default).
`true`: Active sessions of the application are preserved and restored.
If any active session of the application fails to be preserved or restored, none of the sessions will be available when the redeployment is complete. However, the redeployment continues and a warning is logged.
To preserve active sessions, Payara Server serializes the sessions and saves them in memory. To restore the sessions, the class loader of the newly redeployed application deserializes any sessions that were previously saved.
`preserveAppScopedResources`;; If set to `true`, preserves any application-scoped resources and restores them during redeployment. Default is `false`.
`allowFacesCdiInitialisation={false|true}`;; If set to true, enables scanning for Jakarta Faces content in the application, and if detected, forcing an attempt to initialise CDI and Weld even if bean discovery is set to "none". Default is false.

ifeval: ["enterprise" == "community"] `warlibs={false|true}`;; (This property will be overriden by `--warlibs`) If set to true, enables an application to make use of shared libraries installed to `domain_dir/lib/warlibs`. Default is `false`.
Libraries stored in this directory allow developers to create and deploy skinny WARs, as they are subject to additional annotation and extension scanning as well as processing for CDI, EJB, Servlet, and more. Unlike common and application libraries (those stored in

`domain_dir/lib` and `domain_dir/lib/applibs` respectively), warlibs are cached and don't dynamically update if the files within the directory are modified. **endif**: []

Other available properties are determined by the implementation of the component that is being redeployed.
For components packaged as OSGi bundles (`--type=osgi`), the `deploy` subcommand accepts properties arguments to wrap a WAR file as a WAB (Web Application Bundle) at the time of deployment. The subcommand looks for a key named `UriScheme` and, if present, uses the key as a URL stream handler to decorate the input stream. Other properties are used in the decoration process. For example, the Payara Server OSGi web container registers a URL stream handler named `webbundle`, which is used to wrap a plain WAR file as a WAB. For more information about usage, see the example in this help page.

`--skipdsfailure`: This option will skip over a data source failure. This means an application can be deployed with a data source set up, but the database need not be running at the time.

+ Without this option the deployment would fail if a data source is defined and the database is not available at deployment time.

+

```
asadmin deploy --skipdsfailure /home/mark/clusterjsp-1.1.war
```

Operands

file_archive|filepath: The path to the archive that contains the application that is being deployed. This path can be a relative path or an absolute path. The archive can be in either of the following formats: * An archive file, for example, `/export/JEE_apps/hello.war`.
If the `--upload` option is set to `true`, this is the path to the deployable file on the local client machine. If the `--upload` option is set to `false`, this is the path to the file on the server machine. * A directory that contains the exploded format of the deployable

archive. This is the path to the directory on the server machine. If you specify a directory, the `--upload` option is ignored.

Examples

Example 1 Deploying an Enterprise Application

This example deploys the enterprise application packaged in the `Cart.ear` file to the default server instance `server`. You can use the `--target` option to deploy to a different server instance or to a cluster.

```
asadmin> deploy Cart.ear
Application deployed successfully with name Cart.
Command deploy executed successfully
```

Example 2 Deploying a Web Application With the Default Context Root

This example deploys the web application in the `hello.war` file to the default server instance `server`. You can use the `--target` option to deploy to a different server instance or to a cluster.

```
asadmin> deploy hello.war
Application deployed successfully with name hello.
Command deploy executed successfully
```

Example 3 Forcibly Deploying a Web Application With a Specific Context Root

This example forcibly deploys the web application in the `hello.war` file. The context root of the deployed web application is `greetings`. If the application has already been deployed, it is redeployed.

```
asadmin> deploy --force=true --contextroot greetings hello.war
Application deployed successfully with name hello.
Command deploy executed successfully
```

Example 4 Deploying a Connector Module

This example deploys a connector module that is packaged in a RAR file.

This example uses the `--target` option. The target in this example is an existing standalone server instance that does not belong to a cluster.

```
asadmin> deploy --target myinstance jdbcra.rar
Application deployed successfully with name jdbcra.
Command deploy executed successfully
```

Example 5 Specifying the Deployment Order for an Application

This example specifies the deployment order for two applications. The `cart` application is loaded before the `horse` application at server startup.

Some lines of output are omitted from this example for readability.

```
asadmin> deploy --deploymentorder 102 --name cart cart.war
...
asadmin> deploy --deploymentorder 110 --name horse horse.war
...
```

Example 6 Deploying an Application Using an Alternate Jakarta EE 8 Deployment Descriptor File

This example deploys an application using a Jakarta EE standard deployment descriptor file that resides outside the application archive.

```
asadmin> deploy --altdd path_to_alternate_descriptor cart.ear
Application deployed successfully with name cart.
Command deploy executed successfully
```

Example 7 Deploying an Application Using an Alternate Payara Server Deployment Descriptor File

This example deploys an application using a Payara Server runtime deployment descriptor file that resides outside the application archive.

```
asadmin> deploy --runtimealtdd path_to_alternate_runtime_descriptor horse.ear
Application deployed successfully with name horse.
Command deploy executed successfully
```

Example 8 Wrapping a WAR File as a WAB

This example wraps a plain WAR file as a WAB when an OSGi bundle is deployed, and is specific to components packaged as OSGi bundles.

The backslash (\) character is used to escape characters in the command. For more information about escape characters in options for the `asadmin` utility, see the asadmin-1m help page.

```
asadmin deploy --type osgi \
--properties "UriScheme=webbundle:Bundle-SymbolicName=bar:\
Import-Package=javax.servlet;javax.servlet.http;
%20version\\=3.0;resolution\\:
=mandatory:Web-ContextPath=/foo" \
/tmp/test_sample1.war
Application deployed successfully with name sample1.
Command deploy executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get,
- redeploy,
- set,
- undeploy
- "Payara Server Application Deployment section"

Disable-Asadmin-Recorder

Disables the Asadmin recorder.

Synopsis

```
asadmin [asadmin-options] disable-asadmin-recorder [--help]
```

Description

The `disable-asadmin-recorder` subcommand deactivates the Asadmin Recorder feature, ceasing the capture and logging of asadmin commands executed through the Admin Console.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand.

Examples

Example 1 Disabling asadmin Recorder

This example deactivates the Asadmin Recorder feature.

```
asadmin disable-asadmin-recorder

Command disable-asadmin-recorder executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- enable-asadmin-recorder

Disable-Monitoring

Disables monitoring for the server or for specific monitorable module.

Synopsis

```
asadmin [asadmin-options] disable-monitoring [--help]
[--modules module-name][:module-name]*
```

Description

The `disable-monitoring` subcommand is used to turn off monitoring for Payara Server or for particular modules during runtime. Changes are dynamic, that is, server

restart is not required.

Running the `disable-monitoring` subcommand without the `--module` option disables the monitoring service by setting the `monitoring-enabled` attribute of the `monitoring-service` element to `false`. The individual modules retain their monitoring levels, but no monitoring data is generated because the entire monitoring service is disabled.

This subcommand used with the `--modules` option disables monitoring for a module by setting the monitoring level to OFF. The status of the monitoring service is not affected. For a list of monitorable modules, see the `--modules` option in this help page.

An alternative method for disabling monitoring is to use the `set` subcommand. In this case, the server must be restarted for changes to take effect.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--modules`**: Disables the specified module or modules by setting the monitoring level to OFF. Multiple modules are separated by : (colon). Monitorable modules include `connector-connection-pool`, `connector-service`, `ejb-container`, `http-service`, `jdbc-connection-pool`, `jersey`, `jpa`, `jms-service`, `jvm`, `security`, `thread-pool`, `transaction-service`, `web-container`, and `web-services-container`. Additional modules can be listed by using the `get` subcommand.

Operands

Examples

Example 1 Disabling the Monitoring Service for Payara Server

This example disables monitoring for Payara Server in general by setting the `enable-`

`monitoring` flag to `false` (default is `true`).

```
asadmin> disable-monitoring
Command disable-monitoring executed successfully
```

Example 2 Disabling Monitoring for the Web and EJB Containers

This example disables monitoring for specific containers. Their monitoring levels will be set to OFF.

```
asadmin> disable-monitoring --modules web-container:ejb-container
Command disable-monitoring executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get, enable-monitoring,
- list,
- monitor,
- set
- monitoring
- "administering-the-monitoring-service" in Payara Server General Administration section

Disable-Phone-Home

Disables the phone home services.

Synopsis

```
asadmin> disable-phone-home
```

Description

The phone home data is presently used to gather usage statistics about the corresponding distribution, such as how long users typically run instances, and what versions they use. The phone home service is enabled by default when starting a Payara Server and can be disabled by using this command. Make sure to restart Payara Server for the changes to take effect.

Examples

Example 1 Disable Phone Home Service

```
asadmin>  disable-phone-home
Phone Home Service is disabled

Command disable-phone-home executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- enable-phone-home,
- list-phone-home

Disable-Secure-Admin-Internal-User

Instructs the Payara Server DAS and instances to not use the specified admin user to authenticate with each other and to authorize admin operations.

Synopsis

```
asadmin [asadmin-options] disable-secure-admin-internal-user [--help]
admin-username
```

Description

The `disable-secure-admin-internal-user` subcommand disables secure admin from using the username (instead of SSL certificates) to authenticate the DAS and instances with each other and to authorize admin operations.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

admin-username: The admin username that Payara Server should not use to authenticate the DAS and instances with each other and to authorize admin operations.

Examples

Example 1 Disabling a username for secure admin

The following example disables secure admin from using username tester to authenticate the DAS and instances with each other and to authorize admin operations.

```
asadmin> disable-secure-admin-internal-user tester

Command disable-secure-admin-internal-user executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- enable-secure-admin,
- enable-secure-admin-internal-user
- enable-secure-admin-internal-user

Disable-Secure-Admin-Principal

Disables the certificate for authorizing access in secure administration.

Synopsis

```
asadmin [asadmin-options] disable-secure-admin-principal [--help]
--alias aliasname | DN
```

Description

The `disable-secure-admin-principal` subcommand disables the certificate as being valid for authorizing access as part of secure administration.

You must specify either the `--alias` option, or the DN.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--alias`**: The alias name of the certificate in the truststore. Payara Server looks up the certificate in the truststore using that alias and, if found, disables the corresponding DN as being valid for secure administration. Because alias-name must be an alias associated with a certificate currently in the truststore, you may find it most useful for self-signed certificates.

Operands

DN: The distinguished name of the certificate, specified as a comma-separated list in quotes. For example, `"CN=system.amer.oracle.com,OU=GlassFish,O=Oracle Corporation,L=Santa Clara,ST=California,C=US"` .

Examples

Example 1 Disables trust of a DN for secure administration

The following example shows how to disable trust of a DN for authorizing access in secure administration.

```
asadmin> disable-secure-admin-principal
"CN=system.amer.oracle.com,OU=GlassFish,
O=Oracle Corporation,L=Santa Clara,ST=California,C=US"

Command disable-secure-admin-principal executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- enable-secure-admin,
- enable-secure-admin-principal

Disable-Secure-Admin

Disables secure admin if it is already enabled.

Synopsis

```
asadmin [asadmin-options] disable-secure-admin [--help]
```

Description

The `disable-secure-admin` subcommand disables secure admin if it is already enabled.

NOTE	You must restart any running servers in the domain after you enable or disable secure admin. It is simpler to enable or disable secure admin with only the DAS running, then restart the DAS, and then start any other instances.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 Disabling `secure admin` for a domain

The following example shows how to disable `secure admin` for a domain.

```
asadmin> disable-secure-admin
server-config
default-config

Command disable-secure-admin executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- enable-secure-admin

Disable

Disables the component.

Synopsis

```
asadmin [asadmin-options] disable [--help]
[--target target_name] component_name
```

Description

The `disable` subcommand immediately disables the specified deployed component. If the component has not been deployed, an error message is returned.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target on which you are disabling the component. Valid values are:
`server`;; Disables the component on the default server instance `server` and is the default value. domain_name;; Disables the component on the named domain. cluster_name;; Disables the component on every server instance in the cluster. instance_name;; Disables the component on a particular clustered or stand-alone server instance.

Operands

component_name: name of the component to be disabled.
The name can include an optional version identifier, which follows the name and is

separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. To disable multiple versions, you can use an asterisk (`*`) as a wildcard character. For more information about module and application versions, see "module-and-application-versions" in Payara Server Application Deployment section.

Examples

Example 1 Disabling a Component

This example disables the deployed component `sampleApp`.

```
asadmin> disable sampleApp
Command disable executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- deploy,
- enable,
- undeploy
- "Payara Server Application Deployment section"

Enable-Asadmin-Recorder

Enables the asadmin recorder.

Synopsis

```
asadmin [asadmin-options] enable-asadmin-recorder [--help]
```

Description

The `enable-asadmin-recorder` subcommand activates the Asadmin Recorder feature, allowing for the capture and logging of all asadmin commands executed

through the Admin Console.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 Enabling asadmin Recorder

This example activates the Asadmin Recorder feature.

```
asadmin enable-asadmin-recorder

Command enable-asadmin-recorder executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- disable-asadmin-recorder

Enable-Monitoring

Enables monitoring for the server or for specific monitorable modules.

Synopsis

```
asadmin [asadmin-options] enable-monitoring [--help]
[--target target]
[--mbean={false|true}]
[--dtrace={true|false}]
[--modules modules[=level][:module[=level]]*
[--pid pid]
[--options options]]
```

Description

The `enable-monitoring` subcommand is used to turn on monitoring for Payara Server or for particular modules during runtime. Changes are dynamic, that is, server

restart is not required.

By default, the monitoring service is enabled, that is, the `monitoring-enabled` attribute of the `monitoring-service` element is `true`. However, the default monitoring level for individual modules is `OFF`. This subcommand used with the `--modules` option can enable monitoring for a given module by setting the monitoring level to HIGH or LOW. If level is not specified when running the subcommand, the level defaults to HIGH.

The specific meanings of HIGH or LOW are determined by the individual containers. For a list of monitorable modules, see the `--modules` option in this help page.

An alternative method for enabling monitoring is to use the `set` subcommand. In this case, the server must be restarted for changes to take effect.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target on which to enabling monitoring. Valid values are as follows:
`server`;; Enables monitoring for the default server instance `server` and is the default value. configuration-name;; Enables monitoring for the named configuration. cluster-name;; Enables monitoring for every server instance in the cluster. instance-name;; Enables monitoring for a particular server instance. **`--mbean`**: Enables Mbean monitoring. Default value is false. **`--dtrace`**: Only usable if the DTrace Monitoring module is present. Enables Oracle Solaris DTrace monitoring. Default value is false. **`--modules`**: Enables specified module or modules by indicating monitoring level. Valid levels are OFF, HIGH, LOW. If level is not specified, the default setting is HIGH. Multiple modules are separated by : (colon). Monitorable modules include `connector-connection-pool`, `connector-service`, `ejb-container`, `http-service`, `jdbc-connection-pool`, `jersey`, `jpa`, `jms-service`, `jvm`, `security`, `thread-pool`,

`transaction-service`, `web-container`, and `web-services-container`. Additional modules can be listed by using the `get` subcommand. **`--pid`**: Specifies the Payara Server JVM process identifier (PID). When monitoring is enabled, the `btrace-agent` is attached, based on the specified PID. Need to specify only in exceptional cases when the system cannot determine the PID. In this situation, the subcommand prompts for the PID of the corresponding Payara Server process. **`--options`**: Sets the following `btrace-agent` options:
`debug`;; Enables debugging for BTrace. Default value is false.

Examples

Example 1 Enabling the Monitoring Service for Payara Server

This example enables monitoring for Payara Server in general shell setting the `enable-monitoring` flag to `true` (default is `true`)shell

```
asadmin> enable-monitoring
Command enable-monitoring executed successfully
```

Example 2 Enabling Monitoring for the Web and EJB Containers

This example enables monitoring for specific containers by setting their monitoring levels.

```
asadmin> enable-monitoring --modules web-container=LOW:ejb-container=HIGH
Command enable-monitoring executed successfully
```

Example 3 Turning on Debugging for Monitoring

This example turns on debugging.

```
asadmin> enable-monitoring --options debug=true
Command enable-monitoring executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- disable-monitoring,
- get,
- list,
- monitor,
- set
- monitoring
- "administering-the-monitoring-service" in Payara Server General Administration section

Enable-Phone-Home

Enables the phone home services.

Synopsis

```
asadmin> enable-phone-home
```

Description

The phone home data is presently used to gather usage statistics about the corresponding distribution, such as how long users typically run instances, and what versions they use. The phone home service is enabled by default when starting a Payara Server.

Examples

Example 1 Enable Phone Home Service

```
asadmin>  enable-phone-home
Phone Home Service is enabled

Command enable-phone-home executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- disable-phone-home,
- list-phone-home

Enable-Secure-Admin-Internal-User

Instructs the Payara Server DAS and instances to use the specified admin user and the password associated with the password alias to authenticate with each other and to authorize admin operations.

Synopsis

```
asadmin [asadmin-options] enable-secure-admin-internal-user [--help]
[--passwordalias pwdaliasname]
admin-username
```

Description

The `enable-secure-admin-internal-user` subcommand instructs all servers in the domain to authenticate to each other, and to authorize admin operations submitted to each other, using an existing admin username and password rather than SSL certificates. This generally means that you must:

1. Create a valid admin user.

```
asadmin> create-file-user --authrealmname admin-realm --groups
asadmin newAdminUsername
```

2. Create a password alias for the just-created password.

```
asadmin> create-password-alias passwordAliasName
```

3. Use that username and password for inter-process authentication and admin authorization.

```
asadmin> enable-secure-admin-internal-user
--passwordalias passwordAliasName
```

```
newAdminUsername
```

If Payara Server finds at least one secure admin internal user, then if secure admin is enabled Payara Server processes will not use SSL authentication and authorization with each other and will instead use username password pairs.

If secure admin is enabled, all Payara Server processes continue to use SSL encryption to secure the content of the admin messages, regardless of how they authenticate to each other.

Most users who use this subcommand will need to set up only one secure admin internal user. As a general practice, you should not use the same username and password pair for internal admin communication and for admin user login.

If you set up more than one secure admin internal user, you should not make any assumptions about which username and password pair Payara Server will choose to use for any given admin request.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--passwordalias`**: The password alias for the user that Payara Server should use for internally authenticating and authorizing the DAS to instances and the instances to the DAS.

Operands

admin-username: The admin username that Payara Server should use for internally authenticating and authorizing the DAS to instances and the instances to the DAS.

Examples

Example 1 Specifying a username and password for secure admin

The following example allows secure admin to use a username and password alias for authentication and authorization between the DAS and instances, instead of certificates.

```
asadmin> enable-secure-admin-internal-user
--passwordalias passwordAliasName
newAdminUsername
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- disable-secure-admin-internal-user,
- enable-secure-admin

Enable-Secure-Admin-Principal

Instructs Payara Server, when secure admin is enabled, to accept admin requests from clients identified by the specified SSL certificate.

Synopsis

```
asadmin [asadmin-options] enable-secure-admin-principal [--help]
--alias aliasname | DN
```

Description

The `enable-secure-admin-principal` subcommand instructs Payara Server to accept admin requests when accompanied by an SSL certificate with the specified distinguished name (DN). If you use the "`--alias` aliasname" form, then Payara Server looks in its truststore for a certificate with the specified alias and uses the DN associated with that certificate. Otherwise, Payara Server records the value you specify as the DN.

You must specify either the `--alias` option, or the DN.

You can run `enable-secure-admin-principal` multiple times so that Payara Server accepts admin requests from a client sending a certificate with any of the DNs you specify.

When you run `enable-secure-admin`, Payara Server automatically records the DNs for the admin alias and the instance alias, whether you specify those values or use the defaults. You do not need to run `enable-secure-admin-principal` yourself for those certificates. Other than these certificates, you must run `enable-secure-admin-principal` for any other DN that Payara Server should authorize to send admin requests. This includes DNs corresponding to trusted certificates (those with a certificate chain to a trusted authority.)

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--alias`**: The alias name of the certificate in the trust store. Payara Server looks up certificate in the trust store using that alias and, if found, stores the corresponding DN as being valid for secure administration. Because alias-name must be an alias associated with a certificate currently in the trust store, you may find it most useful for self-signed certificates.

Operands

DN: The distinguished name of the certificate, specified as a comma-separated list in quotes. For example, `"CN=system.amer.oracle.com,OU=GlassFish,O=Oracle Corporation,L=Santa Clara,ST=California,C=US"`.

Examples

Example 1 Trusting a DN for secure administration

The following example shows how to specify a DN for authorizing access in secure administration.

```
asadmin> enable-secure-admin-principal
"CN=system.amer.oracle.com,OU=GlassFish,
O=Oracle Corporation,L=Santa Clara,ST=California,C=US"

Command enable-secure-admin-principal executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- disable-secure-admin-principal,
- enable-secure-admin

Enable-Secure-Admin

Enables secure admin (if it is not already enabled), optionally changing the alias used for DAS-to-instance admin messages or the alias used for instance-to-DAS admin messages.

Synopsis

```
asadmin [asadmin-options] enable-secure-admin [--help]
[--adminalias=alias]
[--instancealias=alias]
```

Description

The `enable-secure-admin` subcommand causes the DAS and the instances in the domain to use SSL certificates for encrypting the messages they send to each other. This subcommand also allows the DAS to accept administration messages from remote admin clients such as the asadmin utility and IDEs.

NOTE	You must restart any running servers in the domain after you enable or disable secure admin. It is simpler to enable or disable secure admin with only the DAS running, then restart the DAS, and then start any other instances.

By default, when secure admin is enabled the DAS and the instances use these SSL certificates to authenticate to each other as security "principals" and to authorize admin access. The `--asadminalias` value indicates to the DAS which SSL certificate it should use to identify itself to the instances. The `--instancealias` value determines for instances which SSL certificate they should use to identify themselves to the DAS.

The `enable-secure-admin` subcommand fails if any administrative user in the

domain has a blank password.

Alternatively, you can use the `enable-secure-admin-internal-user` subcommand to cause the servers to identify themselves using a secure admin user name and password.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the link:Technical Documentation/Payara Server Documentation/Command Reference/asadmin#asadmin-1m[`asadmin`] help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--adminalias`**: The alias that refers to the SSL/TLS certificate on the DAS. This alias is used by the DAS to identify itself to instances. The default value is `s1as`. **`--instancealias`**: The alias that refers to the SSL/TLS certificate on the instances. This alias is used by the instances to identify themselves to the DAS. The default value is `glassfish-instance`.

Examples

Example 1 Enabling `secure admin` for a domain

The following example shows how to enable `secure admin` for a domain using an admin alias `adtest` and an instance alias `intest`

```
asadmin> enable-secure-admin --adminalias adtest --instancealias intest
server-config
default-config

Command enable-secure-admin executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- disable-secure-admin,

- enable-secure-admin-principal,
- enable-secure-admin-internal-user

Enable

Enables the component.

Synopsis

```
asadmin [asadmin-options] enable [--help]
[--target target_name] component_name
```

Description

The `enable` subcommand enables the specified deployed component. If the component is already enabled, then it is re-enabled. If it has not been deployed, then an error message is returned.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target on which you are enabling the component. Valid values are:
`server`;; Enables the default server instance `server` and is the default value. domain_name;; Enables the named domain. cluster_name;; Enables every server instance in the cluster. instance_name;; Enables a particular clustered or stand-alone server instance.

Operands

component_name: name of the component to be enabled.
The name can include an optional version identifier, which follows the name and is separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. For more information about module and application versions, see "module-and-application-versions" in Payara Server Application Deployment section.

At most one version of a module or application can be enabled on a server instance. All other versions are disabled. Enabling one version automatically disables all others.

Examples

Example 1 Enabling a Component

This example enables the disabled component, `sampleApp`.

```
asadmin> enable sampleApp
Command enable executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- deploy,
- disable,
- undeploy
- "Payara Server Application Deployment section"

Export-Sync-Bundle

Exports the configuration data of a cluster or standalone instance to an archive file.

Synopsis

```
asadmin [asadmin-options] export-sync-bundle [--help]
--target target
[--retrieve={false|true}]
[file-name]
```

Description

The `export-sync-bundle` subcommand exports the configuration data of a cluster or standalone instance to an archive file. The archive file can then be used with the import-sync-bundle subcommand to restore the configuration data.

Importing an instance's configuration data transfers the data to a host for an instance without the need for the instance to be able to communicate with the domain administration server (DAS). Importing an instance's configuration data is typically required for the following reasons:

- To reestablish the instance after an upgrade
- To synchronize the instance manually with the DAS when the instance cannot contact the DAS

The subcommand creates an archive that contains the following files and directories in the current domain directory:

- All the files in the following directories:
  - `config`
  - `docroot`
- The entire contents of the following directories and their subdirectories:
  - `applications`
  - `config/`target, where target is the cluster or standalone instance for which configuration data is being exported
  - `generated`
  - `lib`

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: The cluster or standalone instance for which to export configuration data. The `--target` option is required. This option must not specify a clustered Payara Server instance. If this option specifies a clustered instance, an error occurs. To export configuration data for a clustered instance, specify the name of the

cluster of which the instance is a member, not the instance. `--retrieve`: Specifies whether the archive file is downloaded from the DAS host to the host where the subcommand is run.
Possible values are as follows:
`true`;; The archive file is downloaded to the host where the subcommand is run.
`false`;; The archive file is not downloaded and remains on the DAS host (default).

Operands

file-name: The file name and location of the archive file to which to export the data.
The default depends on the setting of the `--retrieve` option:
* If `--retrieve` is `false`, the default is `sync/`target`-sync-bundle.zip` in the current domain directory. * If `--retrieve` is `true`, the default is target`-sync-bundle.zip` in the current working directory.
target is the cluster or standalone instance that the `--target` option specifies.
If a relative path is specified, the directory to which the path is appended depends on the setting of the `--retrieve` option:
* If `--retrieve` is `false`, the path is appended to the `config` subdirectory of the current domain directory. * If `--retrieve` is `true`, the path is appended to the current working directory.
If an existing directory is specified without a filename, the file name of the archive file is target`-sync-bundle.zip`, where target is the cluster or standalone instance that the `--target` option specifies.

Examples

Example 1 Exporting the Configuration Data of a Cluster

This example exports the configuration data of the cluster `pmdcluster`.

```
asadmin> export-sync-bundle --target=pmdcluster
Sync bundle: /export/payara7/glassfish/domains/domain1/sync/
pmdcluster-sync-bundle.zip

Command export-sync-bundle executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- import-sync-bundle

Export

Marks a variable name for automatic export to the environment of subsequent commands in multimode.

Synopsis

```
asadmin [asadmin-options] export [--help]
[variable-name=value [variable-name=value]*]
```

Description

In multimode, the `export` subcommand marks an environment variable for automatic export to the environment of subsequent commands. All subsequent commands use the variable name value as specified unless you exit multimode, or use the `unset` subcommand to unset the variable. If only the variable name is specified, the current value of that variable name is displayed.

If the `export` subcommand is used without any arguments, a list of all the exported variables and their values is displayed. Exported shell environment variables set prior to invoking the `asadmin` utility are imported automatically and set as exported variables within `asadmin`. Environment variables that are not exported cannot be read by the`asadmin` utility.

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

variable-name=value: Variable name and value for automatic export to the environment to be used by subsequent commands.

Examples

Example 1 Listing the Environment Variables That Are Set

This example lists the environment variables that have been set.

```
asadmin> export
Command export executed successfully
```

Example 2 Setting an Environment Variable

This example sets the `AS_ADMIN_HOST` environment variable to `bluestar`.

```
asadmin> export AS_ADMIN_HOST=bluestar
Command export executed successfully
```

Example 3 Setting Multiple Environment Variables

This example sets a number of environment variables for the multimode environment.

```
asadmin> export AS_ADMIN_HOST=bluestar AS_ADMIN_PORT=8000
AS_ADMIN_USER=admin AS_ADMIN_PREFIX=server1.jms-service
ceX=server1.jms-service
Command export executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- multimode,
- unset

Flush-Connection-Pool

Reintializes all connections established in the specified connection pool.

Synopsis

```
asadmin [asadmin-options] flush-connection-pool [--help]
[--appname application [--modulename module]
pool_name
```

Description

The `flush-connection-pool` subcommand resets a JDBC connection pool or a connector connection pool to its initial state. Any existing live connections are destroyed, which means that the transactions associated with these connections are lost. The subcommand then recreates the initial connections for the pool, and restores the pool to its steady pool size.

This subcommand is supported in remote mode only.

Application Scoped Resources

The `flush-connection-pool` subcommand can target resources that are scoped to a specific application or module, as defined in the `glassfish-resources.xml` or `payara-resources.xml` for the Payara Server domain.

IMPORTANT	glassfish-resources.xml is deprecated and support will be removed in a future release. It is recommended to use payara-resources.xml instead.

- To reference the `jndi-name` for an application scoped resource, perform the lookup using the `java:app` prefix.
- To reference the `jndi-name` for a module scoped resource, perform the lookup using the `java:module` prefix.

The `jndi-name` for application-scoped-resources or module-scoped-resources are specified using the format `java:app/`jdbc/myDataSource` or `java:module/`jdbc/myModuleLevelDataSource`. This naming scope is defined

in the Jakarta EE Specification (`https://jakarta.ee/specifications/`).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--appname`**: Name of the application in which the application scoped resource is defined. **`--modulename`**: Name of the module in which the module scoped resource is defined.

Operands

pool_name: Name of the connection pool to be reinitialized.

Flushing a Connection Pool

This example reinitializes the JDBC connection pool named `__TimerPool`.

```
asadmin> flush-connection-pool __TimerPool
Command flush-connection-pool executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- list-connector-connection-pools,
- list-jdbc-connection-pools

Flush-Jmsdest

Purges messages in a JMS destination.

Synopsis

```
asadmin [asadmin-options] flush-jmsdest [--help]
--desttype {topic|queue}
[--target target]
destname
```

Description

The `flush-jmsdest` subcommand purges the messages from a physical destination in the server's Java Message Service (JMS) configuration.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--desttype`**: This option indicates the type of physical destination from which you want to purge messages. The supported destination types are `topic` and `queue`. **`--target`**: Purges messages from the physical destination only for the specified target. Valid values are as follows:
`server`;; Purges messages from the physical destination for the default server instance. This is the default value. configuration-name;; Purges messages from the physical destination in the specified configuration. cluster-name;; Purges messages from the physical destination for every server instance in the specified cluster. instance-name;; Purges messages from the physical destination for the specified server instance.

Operands

dest_name: The unique identifier of the JMS destination to be purged.

Examples

Example 1 Purging messages from a physical destination

The following subcommand purges messages from the queue named `PhysicalQueue`.

```
asadmin> flush-jmsdest --desttype queue PhysicalQueue
Command flush-jmsdest executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-jmsdest,
- list-jmsdest

Freeze-Transaction-Service

Freezes the transaction subsystem.

Synopsis

```
asadmin [asadmin-options] freeze-transaction-service [--help]
[--target target]
```

Description

The `freeze-transaction-service` subcommand freezes the transaction subsystem, preventing the transaction manager from starting, completing, or changing the state of all in-flight transactions. Invoke this command before rolling back any in-flight transactions. Invoking this subcommand on an already frozen transaction subsystem has no effect. Restarting the server unfreezes the transaction subsystem. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target on which you are freezing the transaction service. Valid values are:
`server`;; Freezes the transaction service for the default server instance `server` and is the default value. configuration_name;; Freezes the transaction service for all server instances that use the named configuration. cluster_name;; Freezes the transaction service for every server instance in the cluster. instance_name;; Freezes the transaction service for a particular server instance.

Examples

Example 1 Using freeze-transaction-service

```
% asadmin freeze-transaction-service
Command freeze-transaction-service executed successfully
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- recover-transactions,
- rollback-transaction,
- unfreeze-transaction-service
- ""Administering Transactions"" in Payara Server General Administration section
- Transactions" in Jakarta EE Specification

Generate-Bash-Autocomplete

Creates an Asadmin CLI autocompletion script for bash.

Synopsis

```
asadmin generate-bash-autocomplete [--help]
[--force=false]
[--localcommands=false]
[file]
```

Description

The created file will be located in the server's host and not the client. If they are on different hosts, then the file will be needed to copied from the server to the client.

TIP You can add the resulting file to your bash path by adding source `path/to/outputfile` to your .bashrc settings or copy the file into the `/etc/bash_completion.d` directory.

This means that you can run asadmin commands by typing asadmin in a terminal, and you can use double-tab to autocomplete a command, similar to other bash commands.

Options

`--help`: **`-?`**: Displays the help text for the subcommand.

`--force`: Whether to overwrite a script file if one already exists. The default value is `false`.

`--file`: The path for the resulting file. The default value is `as-install/bin/bash_autocomplete`.

`--localCommands`: This option will include local asadmin commands.

+ The default value is `false`.

Examples

Example 1 Creating an asadmin autocompletion script for bash.

This example shows a script that is generated if no file is specified.

```
asadmin generate-bash-autocomplete
Written bash autocomplete file to /export/payara7/glassfish/bin/bash_autocomplete
Command generate-bash-autocomplete executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

Generate-Csr

Generates a new CSR (certificate signing request).

Synopsis

```
asadmin generate-csr [--help]
[--domainName=name]
[--domainDir=directory]
[--node=nodeName]
[--nodeDir=directory]
[--listener=listenername]
[--target=instanceName]
[alias]
```

Description

This command can generate a certificate signing request (CSR) for an instance or listener's self-signed certificate, placing the resultant CSR file in the `as-install/glassfish/tls` directory, using the alias name as the generated file name for the CSR.

IMPORTANT	The command overwrite a CSR with the same name already present in the directory.

Options

`--help`: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are generating the CSR. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--listener`**: The name of the HTTP or IIOP listener to add the certificate to. **`--domainname`**: The name of the domain where the target instance exists.

+ Defaults to `domain1` or the existing domain if only one exists. **`--domainDir`**: The path to the directory containing the target domain.

+ Defaults to `as-install/glassfish/domains` **`--node`**: The name of the node where the target instance exists.

+ Defaults to `localhost-$domainname` **`--nodeDir`**: The path to the directory containing the target node.

+ Defaults to `as-install/glassfish/nodes`

Operands

alias: The alias name to use when generating the CSR for the corresponding self-signed certificate.

Examples

Example 1 Generate a new CSR for the certificate with the `test-cert` alias

```
asadmin generate-csr --listener http-listener-2 --target instance-1 test_cert
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- "SSL Certificate Management",
- "`generate-self-signed-certificate`",
- "`renew-self-signed-certificates`",
- "`add-to-truststore`"

Generate-Encryption-Key

Generates an encryption key for Data Grid secure communication.

Synopsis

```
asadmin [asadmin-options] generate-encryption-key
[--help]
[--dontCheckIfRunning={true|false}]
[--domaindir domaindir]
[domain-name]
```

Description

This command can be used to generate an encryption key used by Payara Server to perform encryption and decryption of the data stored within the Data Grid.

For more information on the Data Grid mechanism, see datagrid-encryption.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--dontCheckIfRunning`**: Determines whether to check if the targeted domain is running or not.

+ The default value is `false`. **`--domaindir`**: The path to the directory containing the target domain. The default domain directory is `as-install/domains`.

Operands

domain-name: The name of the domain to target. The default is `domain1`.

Examples

Example 1 Generates an encryption key.

This example generates an encryption key used by Payara Server to perform encryption and decryption of the data stored within the data grid.

```
asadmin> generate-encryption-key
Enter the current master password>
Command generate-encryption-key executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Domain Data Grid",
- "`list-hazelcast-cluster-members`",
- "`list-hazelcast-members`",
- "`restart-hazelcast`",
- "`list-cache-keys`",
- "`list-caches`",
- "`clear-cache`",
- "`set-hazelcast-configuration`",
- "`get-hazelcast-configuration`"

Generate-Jvm-Report

Shows the JVM machine statistics for a given target instance.

Synopsis

```
generate-jvm-report [--help] [--type=jvm-statistic-type] [--target target]
```

Description

The `generate-jvm-report` subcommand creates a report that shows the threads (dump of stack trace), classes, memory, or loggers for a given target instance, including the domain administration server (DAS). If a type is not specified, a summary report is generated. This subcommand only provides statistics for the Payara Server instance processes. This subcommand provides an alternative to sending Ctrl+Break or `kill -3` signals to Payara Server processes to obtain a stack trace for processes that are hanging.

The information in the report is obtained from managed beans (MBeans) and MXBeans that are provided in the Java Platform, Standard Edition (Java SE) or JDK software with which Payara Server is being used.

If Payara Server is running in the Java Runtime Environment (JRE) software from JDK release 6 or Java SE 6, additional information is provided. For example:

- System load on the available processors
- Object monitors that are currently held or requested by a thread
- Lock objects that a thread is holding, for example, `ReentrantLock` objects and `ReentrantReadWriteLock` objects

If the JRE software cannot provide this information, the report contains the text `NOT_AVAILABLE`.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which you are showing JVM machine statistics.

Valid values are as follows:

`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration. **`--type`**: The type of report that is to be generated. Default is `summary`. `summary`;; Displays summary information about the threads, classes, and memory (default). `memory`;; Provides information about heap and non-heap memory consumption, memory pools, and garbage collection statistics for a given target instance. `class`;; Provides information about the class loader for a given target instance. `thread`;; Provides information about threads running and the thread dump (stack trace) for a given target instance. `log`;; Provides information about the loggers that are registered in the Virtual Machine for the Java platform (Java Virtual Machine or JVM machine)."[Foot1]"

Examples

Example 1 Obtaining Summary Information for the JVM Machine

This example shows a partial listing of a report that is generated if no type is specified. This same report is generated if the `summary` type is specified.

```
asadmin> generate-jvm-report
Operating System Information:
Name of the Operating System: SunOS
Binary Architecture name of the Operating System: sparc, Version: 5.10
Number of processors available on the Operating System: 32
System load on the available processors for the last minute: 7.921875.
(Sum of running and queued runnable entities per minute)
General Java Runtime Environment Information for the VM: 64097@sr1-usca-22
...
sun.desktop = gnome
sun.io.unicode.encoding = UnicodeBig
sun.java.launcher = SUN_STANDARD
sun.jnu.encoding = ISO646-US
sun.management.compiler = HotSpot Client Compiler
sun.os.patch.level = unknown
user.dir = /home/thisuser/GlassFish/glassfishv3/glassfish/domains/mydomain4/config
user.home = /home/thisuser
user.language = en
user.name = thisuser
user.timezone = US/Pacific
Command generate-jvm-report executed successfully
```

Example 2 Obtaining Information for a Particular JVM Machine Type

This example generates a report that shows information on the class loader.

```
asadmin> generate-jvm-report --type=class
Class loading and unloading in the Java Virtual Machine:
Number of classes currently loaded in the Java Virtual Machine: 3,781
Number of classes loaded in the Java Virtual Machine since the startup: 3,868
Number of classes unloaded from the Java Virtual Machine: 87
Just-in-time (JIT) compilation information in the Java Virtual Machine:
Java Virtual Machine compilation monitoring allowed: true
Name of the Just-in-time (JIT) compiler: HotSpot Client Compiler
Total time spent in compilation: 0 Hours 0 Minutes 4 Seconds
Command generate-jvm-report executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-jvm-options,
- delete-jvm-options,
- list-jvm-options

Footnote 1: The terms "Java Virtual Machine" and "JVM" mean a Virtual Machine for the Java platform.

Generate-Self-Signed-Certificate

Generates a new self-signed certificate in the specified server configuration.

Synopsis

```
asadmin generate-self-signed-certificate [--help]
[--reload={false|true}]
[--domainName=name]
[--domainDir=directory]
[--node=nodeName]
[--nodeDir=directory]
[--listener=listenername]
[--target=instanceName]
[--distinguishedName=name]
[--alternativeNames=<"ALT1;ALT2;ALT3">]
[alias]
```

Description

This command can generate a self-signed certificate for an instance, placing the resultant key pair in the target instance or listener's keystore and truststores.

If the instance or listener is configured to use the default keystore and truststore, the command will instead synchronize the instance with the DAS (under the assumption the certificate has been added to the default key and trust store of the DAS), since any certificates added to the instance stores would be lost upon next synchronisation.

NOTE	This command will not overwrite an entry already present in the keystore with the same alias. In this scenario no certificate is generated and the command exits. In the case however where there is not an entry with the same alias in the key store but there is in the trust store, a certificate will be generated and the entry in the trust store will be overwritten.

Options

`--help`: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are generating the certificate. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--reload`**: Whether the HTTP listeners should be reloaded. Defaults to `false` **`--listener`**: The name of the HTTP or IIOP listener to add the certificate to. **`--domainname`**: The name of the domain where the target instance exists.

+ Defaults to `domain1` or the existing domain if only one exists. **`--domainDir`**: The path to the directory containing the target domain.

+ Defaults to `as-install/glassfish/domains` **`--node`**: The name of the node where the target instance exists.

+ Defaults to `localhost-$domainname` **`--nodeDir`**: The path to the directory containing the target node.

+ Defaults to `as-install/glassfish/nodes`

`--distinguishedName`: **`--dn`**: The distinguished name (DN) to use when generating the certificate.

`--alternativeNames`: **`--altnames`**: The semicolon (`;`) separated list of additional Subject Alternative Names to add to the generated certificate.

Operands

alias: The alias name to use when generating the certificate and storing it in the keystore and truststores.

Examples

Example 1 Generate a new self-signed certificate under the `test-cert` alias

```
asadmin generate-self-signed-certificate --dn "CN=test.payara.fish,IP=192.168.1.1"
--listener http-listener-2 --alternativenames
"test2.payara.fish;DNS:test3.payara.fish,IP:127.0.0.1,EMAIL:anon@payara.fish"
--target Instance1 test_cert
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- "SSL Certificate Management",
- "`generate-csr`",
- "`renew-self-signed-certificates`",
- "`add-to-keystore`",
- "`add-to-truststore`"

Get-Active-Module-Config

Displays the current active configuration of a service or instance.

Synopsis

```
asadmin [asadmin-options] get-active-module-config [--help]
```

```
[--target target]
[--all={false|true}]
[service_name]
```

Description

The `get-active-module-config` subcommand displays the current active configuration of a service or instance.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which you want to view the current active configuration of a specific service or an entire instance.
Possible values are as follows:
`server`;; Displays the current active configuration for the default server instance. This is the default value. `domain`;; Displays the current active configuration for the default domain. cluster-name;; Displays the current active configuration for every server instance in the specified cluster. instance-name;; Displays the current active configuration for the specified instance. **`--all`**: Displays all current active configurations.
The default value is `false`.

Operands

service_name: The name of the module for which you want to display the current active configuration.

Examples

Example 1 Displaying the Current Active Configuration

This example displays the current active configuration for the JMS service in `server-config` (the default configuration).

```
asadmin> get-active-module-config jms-service
At location: domain/configs/config[server-config]
```

```
<jms-service default-jms-host="default_JMS_host" type="EMBEDDED"
  <jms-host port="7676" host="localhost" name="default_JMS_host"/>
</jms-service>
Command get-active-module-config executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- create-module-config,
- delete-module-config

Get-Admin-Audit-Configuration

Retrieves the current settings of the Admin Audit Service.

Synopsis

```
asadmin [asadmin-options] get-admin-audit-configuration [--help]
[--target=server]
```

Description

The `get-admin-audit-configuration` subcommand is used to list the configuration settings of the auditing service.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the server instance or cluster for which to display the current admin audit configuration.

Examples

Example

```
asadmin> get-admin-audit-configuration

Enabled  Audit Level
false    MODIFIERS
```

```
Name      Notifier Enabled
LOG       false

Command get-admin-audit-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-admin-audit-configuration,
- enable-asadmin-recorder,
- disable-asadmin-recorder,
- asadmin-recorder-enabled

Get-Asadmin-Recorder-Configuration

Returns the current configuration of the Asadmin Recorder service.

Synopsis

```
asadmin [asadmin-options] get-asadmin-recorder-configuration [--help]
```

Description

The `get-asadmin-recorder-configuration` command is used to retrieve the current configuration settings of the Asadmin Recorder feature in Payara Server.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1

Returns the current configuration of the asadmin recorder.

```
asadmin> get-asadmin-recorder-configuration
Enabled  Filter Commands  Output Location                  Filtered Commands
Prepend Enabled  Prepended Options
true     true             /home/user/asadmin-commands.txt  version,_(.*),list(.*
),get(.*),(.*)-list-services,uptime,enable-asadmin-recorder,disable-asadmin-
recorder,set-asadmin-recorder-configuration,asadmin-recorder-enabledfalse
Command get-asadmin-recorder-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-asadmin-recorder-configuration,
- enable-asadmin-recorder,
- disable-asadmin-recorder,
- asadmin-recorder-enabled,

Get-Aws-Config-Source-Configuration

Gets the configuration for the AWS Secrets configuration source.

Synopsis

```
asadmin [asadmin-options] get-aws-config-source-configuration [--help]
[--target=server]
```

Description

The `get-aws-config-source-configuration` retrieves the current configuration for the AWS Config Source.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--target`: Specifies the target where the configuration source resides.

Valid values are as follows:

`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Getting the configuration of the AWS secrets

This example retrieves the AWS Secrets Cloud Config Source.

```
asadmin get-aws-config-source-configuration

Enabled  Region Name  Secret Name
true     eu-west-2    payara/test/key
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-aws-config-source-configuration

Get-Azure-Config-Source-Configuration

Gets the configuration settings of an Azure Secrets configuration source.

Synopsis

```
asadmin [asadmin-options] get-azure-config-source-configuration [--help] [--target
=server]
```

Description

The `get-azure-config-source-configuration` retrieves the current configuration for the Azure Config Source.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the

subcommand.

`--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Getting the configuration of the Azure secrets

This example retrieves the Azure Secrets Cloud Config Source.

```
asadmin> get-azure-config-source-configuration

Enabled Tenant ID Client ID Key VaultName Private Key Path Thumbprint
true 22b3bb26-e046-42df-9c96-65dbd72c1c81 22b3bb26-e046-42df-9c96-65dbd72c1c81
demo-secret-key-vault path/to/privatekey.pem
84E05C1D98BCE3A5421D225B140B36E86A3D5534
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-azure-config-source-configuration

Get-Cdieventbus-Notifier-Configuration

Retrieves the settings of the Event Bus notifier

Synopsis

```
asadmin [asadmin-options] get-eventbus-notifier-configuration [--help]
[--target target]
```

Description

This command can be used to retrieves the current settings of the Event Bus notifier in Payara Server.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the settings of the CDI Event Bus notifier. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance.

Examples

Example 1 Retrieve the settings for the CDI Event Bus notifier in the DAS

```
asadmin get-cdieventbus-notifier-configuration

Enabled  Filter   Loopback
false    WARNING  false
Command get-cdieventbus-notifier-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-cdieventbus-notifier-configuration

Get-Client-Stubs

Retrieves the application JAR files needed to launch the application client.

Synopsis

```
asadmin [asadmin-options] get-client-stubs [--help]
--appname application_name
local_directory_path
```

Description

The `get-client-stubs` subcommand copies the required JAR files for an `AppClient` standalone module or each `AppClient` module in an application from the

server machine to the local directory. Each client's generated JAR file is retrieved, along with any required supporting JAR files . The client JAR file name is of the form app-name`Client.jar`. Before executing the `get-client-stubs` subcommand, you must deploy the application or module. The generated client JAR file is useful for running the application using the `appclient` utility. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--appname`**: The name of the application or stand-alone client module. The name can include an optional version identifier, which follows the name and is separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. For more information about module and application versions, see "module-and-application-versions" in Payara Server Application Deployment section.

Operands

local_directory_path: The path to the local directory where the client stub JAR file should be stored.

Examples

Example 1 Using get-client-stubs

```
asadmin> get-client-stubs --appname myapplication /sample/example
Command get-client-stubs executed successfully
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- deploy,

- redeploy,
- undeploy
- appclient,
- package-appclient
- "Payara Server Application Deployment section"

Get-Config-Dir

Gets the settings of the Directory configuration settings.

Synopsis

```
asadmin [asadmin-options] get-config-dir [--help] [--target
=<target[default:server]>]
```

Description

The `get-config-dir` gets the value of the directory to be used for the directory config source.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target where the configuration source resides. Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Getting the Config Directory.

This example gets the directory to be used for the directory config source.

```
asadmin get-config-dir --target=myAppCluster
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-config-dir

Get-Config-Ordinal

Returns the ordinal value for the given configuration source type.

Synopsis

```
asadmin [asadmin-options] get-config-ordinal [--help] [--target=server]
[--source={domain|config|server|application|module|cluster}]
```

Description

The `get-config-ordinal` returns ordinal value for the given ConfigSource type.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--source`**: The ConfigSource to get the ordinal for.

+ Must be one of: `domain`, `config`, `server`, `application`, `module`, `cluster`
`--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Getting the ordinal value

This example gets the ordinal value for the source 'cluster'.

```
asadmin> get-config-ordinal --source=cluster
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-config-ordinal

Get-Config-Property

Provides a way to get the value of a MicroProfile configuration property stored in a specific configuration source.

Synopsis

```
asadmin [asadmin-options] get-config-property [--help]
[--propertyName=<property.name>]
[--source={domain|config|server|application|module|cluster}]
[--sourceName=<source.name>]
[--moduleName=<module.name>]
[--target=<target[default:server]>]
```

Description

The `get-config-property` gets the value for the given Eclipse MicroProfile configuration property name in one of the built-in configuration sources.

The source is specified with `--source` and, where there is ambiguity, the `--sourceName` and `--moduleName` options can be used.

For example, where the source is `server`, the `--sourceName` can be used to specify the name of the server where the property is to be stored.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--propertyName`: The name of the configuration property to get. `--source`: The configuration source where the property is stored. `--sourceName`: The name of the source when there may be ambiguity, for example a source of type `application` must specify the name of the application.

+ This property is required for sources of type: `config`, `server`, `application` or `module` `--moduleName`: The name of the module when the source is of type `module`. When this is specified, the `sourceName` parameter must be provided and must have the name of the application where the module is deployed. `--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Getting the configuration property value

This example gets the `JMSBrokerURL` property value within the configuration source named `myApplication`, belonging to the module `myModule`, and targeting the `myAppCluster` cluster.

```
asadmin get-config-property --propertyName=JMSBrokerURL --source=module
--sourceName=myApplication --moduleName=myModule --target=myAppCluster
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-config-property
- set-config-property

Get-Datadog-Notifier-Configuration

Retrieves the settings of the DataDog notifier.

Synopsis

```
asadmin [asadmin-options] get-datadog-notifier-configuration [--help]
[--target target]
```

Description

This command can be used to retrieves the current settings of the DataDog notifier in Payara Server.

IMPORTANT	This command is available as long as the DataDog notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the settings of the DataDog notifier. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to on a specified server instance.

Examples

Example 1 Retrieve the settings for the DataDog notifier in the DAS

```
asadmin get-datadog-notifier-configuration

Enabled  Filter   Key
true     WARNING  af510f122026bc9107f7c5f4f049e597
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-datadog-notifier-configuration

Get-Discord-Notifier-Configuration

Retrieves the settings of the Discord notifier.

Synopsis

```
asadmin [asadmin-options] get-discord-notifier-configuration [--help]
[--target target]
```

Description

This command can be used to retrieves the current settings of the Discord notifier in Payara Server.

IMPORTANT	This command is available as long as the Discord notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the settings of the Discord notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to on a specified server instance.

Examples

Example 1 Retrieve the settings for the Discord notifier in the DAS

```
asadmin get-discord-notifier-configuration
```

```
Enabled  Filter   Webhook ID  Webhook Token
true     WARNING  xxx         yyy
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-discord-notifier-configuration

Get-Dynamodb-Config-Source-Configuration

Gets the configuration for the DynamoDB configuration source.

Synopsis

```
asadmin [asadmin-options] get-dynamodb-config-source-configuration [--help]
[--target=server]
```

Description

The `get-dynamodb-config-source-configuration` retrieves the current configuration for the DynamoDB Config Source.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Getting the configuration of the DynamoDB secrets

This example retrieves the current configuration for the DynamoDB Config Source.

```
asadmin get-dynamodb-config-source-configuration
Enabled  Region Name  Table Name  Key Column Name  Value Column Name  Limit
true     eu-west-2    Books       ISBN             Title              10
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-dynamodb-config-source-configuration

Get-Ejb-Invoker-Configuration

Retrieves the `ejb-invoker` application settings.

Synopsis

```
asadmin [asadmin-options] set-ejb-invoker-configuration [--help]
[--target] target
```

Description

Returns the current configuration options for the `ejb-invoker` application on the targeted configuration.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which retrieve the EJB invoker endpoint's configuration. Valid values are as follows:
`server`;; Retrieves the configuration for the EJB invoker endpoint configured for the default server instance `server` and is the default value. cluster-name;; Retrieves the configuration for the EJB invoker endpoint configured for every server instance in the cluster. deployment-group-name;; Retrieves the configuration for the EJB invoker endpoint configured for every server instance in the deployment group. instance-name;;

Retrieves the configuration for the EJB invoker endpoint configured for a particular server instance.

Examples

The following command returns the current configuration options from the deployment group `test-dg`:

```
asadmin> get-ejb-invoker-configuration --target test-dg
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-ejb-invoker-configuration

Get-Email-Notifier-Configuration

Retrieves the settings of the Email notifier

Synopsis

```
asadmin [asadmin-options] get-email-notifier-configuration [--help]
[--target target]
```

Description

This command can be used to retrieves the current settings of the Email notifier in Payara Server.

IMPORTANT	This command is available as long as the Email notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options,

see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are retrieving the settings of the Email notifier. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to on a specified server instance.

Examples

Example 1 Retrieve the settings for the Email notifier in the DAS

```
asadmin get-email-notifier-configuration

Enabled  JNDI Name                  Recipient
true     mail/sampleSession         notifications@example.com
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-email-notifier-configuration

Get-Environment-Warning-Configuration

Retrieves the current settings of the Environment Warning feature in the Admin Console

Synopsis

```
asadmin [asadmin-options] get-environment-warning-configuration [--help]
```

Description

The `get-environment-warning-configuration` subcommand gets the current environment warning configuration.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the

subcommand.

Examples

Example

```
asadmin> get-environment-warning-configuration
Enabled  Message                          Background Colour  Text Colour
true     Caution, this is on production  #ff00ff            #00ff00
Command get-environment-warning-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-environment-warning-configuration

Get-Eventbus-Notifier-Configuration

Retrieves the settings of the Event Bus notifier

Synopsis

```
asadmin [asadmin-options] get-eventbus-notifier-configuration [--help]
[--target target]
```

Description

This command can be used to retrieves the current settings of the Event Bus notifier in Payara Server.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the settings of the Event Bus notifier. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified

server instance.

Examples

Example 1 Retrieve the settings for the Event Bus notifier in the DAS

```
asadmin get-eventbus-notifier-configuration

Enabled  Filter   Topic Name
false    WARNING  payara.notification.event

Command get-eventbus-notifier-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-eventbus-notifier-configuration

Get-Fault-Tolerance-Configuration

Retrieves the configuration settings of the MicroProfile Fault Tolerance service.

Synopsis

```
asadmin [asadmin-options] get-fault-tolerance-configuration [--help]
[--target=server-config]
```

Description

The `get-fault-tolerance-configuration` returns the current configuration options for the MicroProfile Fault Tolerance service on the specified target.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--target`: Specifies the target of this command.
Valid values are as follows:

`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Getting MicroProfile Fault Tolerance configuration

This example gets the current configuration of the Fault Tolerance service.

```
asadmin get-fault-tolerance-configuration --target=instance1
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-fault-tolerance-configuration

Get-Gcp-Config-Source-Configuration

Gets the configuration for the GCP configuration source.

Synopsis

```
asadmin [asadmin-options] get-gcp-config-source-configuration [--help]
[--target=server]
```

Description

The `get-gcp-config-source-configuration` retrieves the current configuration for the GCP Config source.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--target`: Specifies the target where the configuration source resides.

Valid values are as follows:

`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Getting the configuration of the GCP secrets

This example retrieves the current configuration for the GCP Config Source.

```
asadmin> get-gcp-config-source-configuration

Enabled  Project Name  JSON Key File
true     payara-test   payara-test-8a8aa4fbb520.json
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-gcp-config-source-configuration

Get-Hashicorp-Config-Source-Configuration

Gets the configuration for the HashiCorp configuration source.

Synopsis

```
asadmin [asadmin-options] get-hashicorp-config-source-configuration [--help]
[--target=server]
```

Description

The `get-hashicorp-config-source-configuration` retrieves the current configuration for the HashiCorp Config Source.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the

subcommand.

`--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Getting the configuration of the HashiCorp secrets

This example retrieves the current configuration for the HashiCorp Config Source.

```
asadmin get-hashicorp-config-source-configuration

Enabled  API Version  Secrets Engine Path  Secrets Path  Vault Address
true     2            secret               my-secret     https://127.0.0.1:8200
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-hashicorp-config-source-configuration

Get-Hazelcast-Configuration

Retrieves the current Data Grid (Hazelcast) configuration.

Synopsis

```
asadmin [asadmin-options] get-hazelcast-configuration
[--help]
[--target target]
[--checkEncrypted={true|false}]
```

Description

This command can be used to retrieve the current Data Grid (Hazelcast) configuration settings.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are executing this operation. Valid values are:
`server`;; Applies to the default server instance. This is the default value.
instance_name;; Applies to a specified server instance. **`--checkEncrypted`**: This option helps check if encryption of data stored in the "Domain Data Grid" store is enabled or disabled.

Examples

Example 1 Retrieves the current Hazelcast configuration.

```
asadmin> get-hazelcast-configuration
Property Name                          PropertyValue            Scope
Configuration File                     hazelcast-config.xml     Domain
Interfaces                                                      Domain
Auto Increment Port                    false                    Domain
Start Port                             5902                     Domain
Cluster Name                           development              Domain
License Key                                                     Domain
Host Aware Partitioning                true                     Domain
DAS Public Address                                              Domain
DAS Bind Address                                                Domain
DAS Port                               4900                     Domain
Cluster Mode                           multicast                Domain
Tcpip Members                          127.0.0.1:5900           Domain
DNS Members                                                     Domain
MulticastGroup                         224.2.2.3                Domain
MulticastPort                          6666                     Domain
Kubernetes Namespace                                            Domain
Kubernetes Service Name                                         Domain
Encrypt Datagrid                       false                    Domain
Enabled                                true                     Config
JNDIName                               payara/Hazelcast         Config
Cache Manager JNDI Name                payara/CacheManager      Config
Caching Provider JNDI Name             payara/CachingProvider   Config
Lite Member                            false                    Config
Member Name                            payara                   Config
Member Group                           MicroShoal               Config
Executor Pool Size                     4                        Config
Executor Queue Capacity                20                       Config
Scheduled Executor Pool Size           4                        Config
Scheduled Executor Queue Capacity      20                       Config
Public Address                                                  Config
Config Specific Data Grid Start Port                            Config
Command get-hazelcast-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Domain Data Grid",
- "`list-hazelcast-cluster-members`",
- "`list-hazelcast-members`",
- "`restart-hazelcast`",
- "`list-cache-keys`",
- "`list-caches`",
- "`clear-cache`",
- "`set-hazelcast-configuration`",
- "`generate-encryption-key`"

Get-Health

Provides information on the cluster health.

Synopsis

```
asadmin [asadmin-options] get-health [--help]
cluster_name
```

Description

The `get-health` subcommand gets information about the health of the cluster. Note that if the group management service (GMS) is not enabled, the basic information about whether the server instances in this cluster are running or not running is not returned. For each server instance, one of the following states is reported: `not started`, `started`, `stopped`, `rejoined`, or `failed`. This subcommand is available in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

cluster_name: The name of the cluster for which you want the health information. This subcommand prompts you for the cluster name if you don't specify it.

Examples

Example 1 Checking the health of server instances in a cluster

```
asadmin> get-health cluster1
instance1 started since Wed Sep 29 16:32:46 EDT 2010
instance2 started since Wed Sep 29 16:32:45 EDT 2010
Command get-health executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- validate-multicast

Get-Healthcheck-Configuration

Lists the current Healthcheck service settings.

Synopsis

```
asadmin [asadmin-options] get-healthcheck-configuration
[--help]
[--target target]
```

Description

Lists the current configuration for the health check service, configured checkers and enabled notifiers.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the Healthcheck service settings. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance.

Examples

Example 1 Retrieves configuration in the DAS

A sample output is as follows:

```
asadmin get-healthcheck-configuration

Health Check Service Configuration is enabled?: true
Historical Tracing Enabled?: true
Historical Tracing Store Size: 20

Name                   Notifier Enabled
log-notifier           true
jms-notifier           false
cdieventbus-notifier   false
eventbus-notifier      false

Below are the list of configuration details of each checker listed by its name.

Name  Enabled  Time  Unit     Add to MicroProfile Health  Critical Threshold
Warning Threshold  Good Threshold
CPUC  true     5     MINUTES  true                        80                 50
0
HEAP  true     5     MINUTES  false                       80                 50
0

Name   Enabled  Time  Unit     Add to MicroProfile Health  Threshold Time
Threshold Unit
STUCK  true     5     MINUTES  false                       5              MINUTES

Name  Enabled  Time  Unit     Add to MicroProfile Health
MPM   true     5     MINUTES  false

Monitored Metric Name  Description
base_thread_max_count Displays the peak live thread count since the Java virtual
machine started or peak was reset. This includes daemon and non-daemon threads.
base_gc_total_total    Displays the total number of collections that have occurred.
This attribute lists -1 if the collection count is undefined for this collector.

Command get-healthcheck-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "`set-healthcheck-service-configuration`",
- "`list-healthcheck-services`"

Get-Jdbc-Config-Source-Configuration

Gets the settings of the JDBC configuration source.

Synopsis

```
asadmin [asadmin-options] get-jdbc-config-source-configuration [--help] [--target
=server]
```

Description

The `get-jdbc-config-source-configuration` retrieves the current configuration for the JDBC Config Source

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target where the configuration source resides. Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster.

Example

Example 1 Retrieving the JDBC Config Source configurations.

This example gets the configuration of the JDBC config source.

```
asadmin get-jdbc-config-source-configuration
JNDI Name  Table Name  Key-Column Name  Value-Column Name
jndi-name  Books       ISBN             Title
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-jdbc-config-source-configuration

Get-Jms-Notifier-Configuration

Retrieves the settings of the JMS notifier

Synopsis

```
asadmin [asadmin-options] get-log-notifier-configuration [--help]
[--target target]
```

Description

This command can be used to retrieves the current settings of the JMS notifier in Payara Server Full Profile.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the settings of the JMS notifier. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to on a specified server instance.

Examples

Example 1 Retrieve the settings for the JMS notifier in the DAS

```
asadmin get-jms-notifier-configuration

Enabled  Filter   Context Factory Class  Connection Factory Name  Queue Name  URL
Username  Password
true     WARNING  com.sun.enterprise.naming.SerialInitContextFactory
jms/__defaultConnectionFactory  notifierQueue  localhost:7676  myusername
mypassword
```

```
Command get-jms-notifier-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-jms-notifier-configuration

Get-Jmx-Monitoring-Configuration

Retrieves the JMX monitoring service general settings.

Synopsis

```
asadmin [asadmin-options] get-jmx-monitoring-configuration
[--help]
[--target target]
[--pretty={false|true}]
```

Description

This command can be used to view the current configuration of the JMX monitoring service.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the JMX monitoring service configuration. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`pretty`**: Default to `false`. Whether to displays boolean configuration values options as a **tick** or a **cross**

Examples

Example 1 Get current JMX monitoring service configuration

```
asadmin get-jmx-monitoring-configuration
JMX Monitoring Enabled  JMX Log Frequency  JMX Log Frequency Unit
true                    5                  MINUTES

Name                   Notifier Enabled
slack-notifier         false
snmp-notifier          false
teams-notifier         false
datadog-notifier       false
jms-notifier           false
newrelic-notifier      false
log-notifier           true
cdieventbus-notifier   false
eventbus-notifier      false
email-notifier         false
discord-notifier       false

Object Name             Attribute             Description
java.lang:type=Memory   HeapMemoryUsage.used
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Monitoring Basics",
- "`set-amx-enabled`",
- "`set-jmx-monitoring-configuration`"

Get-Ldap-Config-Source-Configuration

Gets the settings of the LDAP configuration source.

Synopsis

```
asadmin [asadmin-options] get-ldap-config-source-configuration [--help] [--target
=server]
```

Description

The `get-ldap-config-source-configuration` gets the current configuration settings for the LDAP configuration source.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target where the configuration source resides. Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster.

Example

Example 1 Getting the LDAP configuration

This example retrieves the current configuration for the LDAP Config Source.

```
asadmin get-ldap-config-source-configuration
Enabled     Url                              Auth Type       Start TLS Enabled
BindDN                                  BindDN Password     Search Base     Search
Filter       Search Scope        Connection Timeout      Read Timeout
true        ldap://ldap.forumsys.com:389    simple          false
cn=read-only-admin,dc=example,dc=com    password
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-ldap-config-source-configuration

Get-Log-Notifier-Configuration

Retrieves the settings of the Log notifier

Synopsis

```
asadmin [asadmin-options] get-log-notifier-configuration [--help]
[--target target]
```

Description

This command can be used to retrieves the current settings of the log notifier in Payara

Server.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the settings of the log notifier. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance.

Examples

Example 1 Retrieve the settings for the log notifier in the DAS

```
asadmin> get-log-notifier-configuration

Enabled  Filter   Use Separate Log File
true     WARNING  false
Command get-log-notifier-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-log-notifier-configuration,

Get-Metrics-Configuration

Gets the configuration settings of the MicroProfile Metrics service.

Synopsis

```
asadmin [asadmin-options] get-metrics-configuration [--help]
[--target <instance-name>]
```

Description

The `get-metrics-configuration` returns the current configuration options for the

metrics service on the targeted configuration.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--target`: Specifies the target for its respective configuration.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Getting the configuration of the metrics service

This example get the metrics service configuration of the `dg1` deployment group

```
asadmin get-metrics-configuration --target dg1
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-metrics-configuration

Get-Microprofile-Healthcheck-Configuration

Retrieves the configuration settings of the MicroProfile HealthCheck service.

Synopsis

```
asadmin [asadmin-options] get-microprofile-healthcheck-configuration [--help]
[--target=server-config]
```

Description

The `get-microprofile-healthcheck-configuration` returns the current configuration options for the MicroProfile Health Check service on the specified target.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--target`: Specifies the target of this command.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Getting MicroProfile Health Check configuration

This example gets the current configuration of the Health Check service.

```
asadmin get-microprofile-healthcheck-configuration --target=instance1

Enabled  EndPoint  VirtualServers  Security Enabled  Roles
true     health                   false             microprofile
Command get-microprofile-healthcheck-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-microprofile-healthcheck-configuration

Get-Monitoring-Level

Retrieves the level configuration of a monitoring service's server module.

Synopsis

```
asadmin [asadmin-options] get-monitoring-level
[--help]
[--module module]
[--target target]
```

Description

This command can be used to get monitoring level of a module.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are retrieving the level configuration of a monitored module. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. `--module`: The name of the module. If omitted, the command will return the configuration of all registered modules.

Examples

Example 1 Get the monitoring level set for the JVM Module

```
asadmin get-monitoring-level --module=jvm --target=server

Module  Monitoring Level
jvm     HIGH
Command get-monitoring-level executed successfully.
```

Example 2 Get the monitoring level set for all modules

```
asadmin get-monitoring-level --target=server

Module                       Monitoring Level
cloud                        OFF
cloud-elasticity             OFF
cloud-orchestrator           OFF
cloud-tenant-manager         OFF
cloud-virt-assembly-service  OFF
connector-connection-pool    OFF
connector-service            OFF
deployment                   OFF
ejb-container                OFF
http-service                 LOW
```

```
jdbc-connection-pool          OFF
jersey                        OFF
jms-service                   OFF
jpa                           OFF
jvm                           HIGH
orb                           OFF
security                      OFF
thread-pool                   OFF
transaction-service           OFF
web-container                 OFF
web-services-container        OFF

Command get-monitoring-level executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Monitoring Basics",
- "`get-monitoring-service-configuration`"
- "`set-monitoring-level`"

Get-Monitoring-Service-Configuration

Retrieves the monitoring service general settings.

Synopsis

```
asadmin [asadmin-options] get-monitoring-service-configuration
[--help]
[--target target]
```

Description

This command can be used to view the current configuration of the monitoring service.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the monitoring service configuration. Valid values are:

`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance.

Examples

Example 1 Get current monitoring service configuration

```
asadmin get-monitoring-service-configuration
Monitoring Enabled  AMX Enabled  MBeans Enabled  DTrace Enabled
true                true         true            false

Command get-monitoring-service-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Monitoring Basics",

Get-Newrelic-Notifier-Configuration

Retrieves the settings of the New Relic notifier.

Synopsis

```
asadmin [asadmin-options] get-newrelic-notifier-configuration [--help]
[--target target]
```

Description

This command can be used to retrieves the current settings of the New Relic notifier in Payara Server.

IMPORTANT	This command is available as long as the New Relic notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the settings of the New Relic notifier. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to on a specified server instance.

Examples

Example 1 Retrieve the settings for the New Relic notifier in the DAS

```
asadmin get-newrelic-notifier-configuration

Enabled  Filter  Key                                Account Id
true     WARNING   b5815wdxj6lF_tmMBljQa5y1603JTiLh  1658989
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-newrelic-notifier-configuration

Get-Notification-Configuration

Retrieves the notification service general settings.

Synopsis

```
asadmin [asadmin-options] get-notification-configuration
[--help]
[--target target]
```

Description

This command can be used to view the current configuration of the notification service.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the notification service configuration. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance.

Examples

Example 1 Get current notification service configuration

```
asadmin get-notification-configuration

Notifier Name          Enabled  Notifier Enabled  Filter Level
slack-notifier         true     false             WARNING
snmp-notifier          true     false             WARNING
teams-notifier         true     false             WARNING
datadog-notifier       true     false             WARNING
jms-notifier           true     false             WARNING
newrelic-notifier      true     false             WARNING
log-notifier           true     true              WARNING
cdieventbus-notifier   true     false             WARNING
eventbus-notifier      true     false             WARNING
email-notifier         true     false             WARNING
discord-notifier       true     false             WARNING

Command get-notification-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-log-notifier-configuration,
- get-jms-notifier-configuration,
- get-eventbus-notifier-configuration,
- get-cdieventbus-notifier-configuration

Get-Openapi-Configuration

Gets the current settings of the MicroProfile OpenAPI service.

Synopsis

```
asadmin [asadmin-options] get-openapi-configuration [--help]
  [--target=<target>]
```

Description

This command get the current settings of the MicroProfile OpenAPI service.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target of this command.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-openapi-configuration

Get-Requesttracing-Configuration

Retrieves the Request Tracing service's configuration.

Synopsis

```
asadmin [asadmin-options] get-requesttracing-configuration [--help]
```

Description

This command can be used to list the configuration settings of the Request Tracing.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand.

Examples

Example 1 Retrieve Request Tracing Service Configuration

```
asadmin> get-requesttracing-configuration
```

Will give an output similar to the following:

```
Request Tracing Service Enabled?: true
Request Tracing Service Sample Rate: 1.0
Request Tracing Service Adaptive Sampling Enabled?: false
Request Tracing Service Application Only?: true
Request Tracing Service Threshold Value: 30
Request Tracing Service Threshold Unit: SECONDS
Request Tracing Service Sample Rate First?: true
Request Tracing Service Reservoir Sampling Enabled?: false
Request Tracing Service Trace Store Size: 20
Request Tracing Service Historic Trace Store Enabled?: false
Below are the configuration details of each notifier listed by its name.

Notifier Name         Notifier Enabled
jms-notifier          false
log-notifier          true
cdieventbus-notifier  false
eventbus-notifier     false
Command get-requesttracing-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-requesttracing-configuration
- list-requesttraces

Get-Rest-Monitoring-Configuration

Retrieves the REST monitoring service general settings.

Synopsis

```
asadmin [asadmin-options] get-rest-monitoring-configuration
[--help]
[--target target]
```

Description

This command can be used to view the current configuration of the JMX monitoring service.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the REST monitoring service configuration. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance.

Examples

Example 1 Get current REST monitoring service configuration

```
asadmin get-rest-monitoring-configuration
Rest Monitoring Enabled  Rest Monitoring Application Name  Context Root
Security Enabled
false                    __restmonitoring                 /rest-monitoring  false
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Monitoring Basics",
- "`set-rest-monitoring-configuration`"

Get-Slack-Notifier-Configuration

Retrieves the settings of the Slack notifier.

Synopsis

```
asadmin [asadmin-options] get-slack-notifier-configuration [--help]
[--target target]
```

Description

This command can be used to retrieves the current settings of the Slack notifier in Payara Server.

IMPORTANT	This command is available as long as the Slack notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the settings of the Slack notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to on a specified server instance.

Examples

Example 1 Retrieve the settings for the Slack notifier in the DAS

```
asadmin get-slack-notifier-configuration

Enabled  Filter   Token 1    Token 2    Token 3
true     WARNING  T43CHENUC  B441EFQB1  Jas8P3jCZ4wK8Y2m88dCxGkn
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-slack-notifier-configuration

Get-Snmp-Notifier-Configuration

Retrieves the settings of the SNMP notifier

Synopsis

```
asadmin [asadmin-options] get-snmp-notifier-configuration [--help]
[--target target]
```

Description

This command can be used to retrieves the current settings of the SNMP notifier in Payara Server.

IMPORTANT	This command is available as long as the SNMP notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the settings of the SNMP notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to on a specified server instance.

Examples

Example 1 Retrieve the settings for the SNMP notifier in the DAS

```
asadmin > get-snmp-notifier-configuration
```

```
Enabled  Filter   Community  OID                Version  Host       SNMP Port
true     WARNING  example    .1.3.6.1.2.1.1.8   v2c      127.0.0.1  162
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-snmp-notifier-configuration

Get-Teams-Notifier-Configuration

Retrieves the settings of the MS Teams notifier.

Synopsis

```
asadmin [asadmin-options] get-teams-notifier-configuration [--help]
[--target target]
```

Description

This command can be used to retrieves the current settings of the MS Teams notifier in Payara Server.

IMPORTANT	This command is available as long as the MS Teams notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are retrieving the settings of the MS Teams notifier. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to on a specified server instance.

Examples

Example 1 Retrieve the settings for the MS Teams notifier in the DAS

```
asadmin get-teams-notifier-configuration

Enabled  Filter   Webhook URL
true     WARNING  https://outlook.office.com/webhook/xyz
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-teams-notifier-configuration

Get-Toml-Config-Source-Configuration

Gets the settings of the TOML configuration source.

Synopsis

```
asadmin [asadmin-options] get-toml-config-source-configuration [--help] [--target
=server]
```

Description

The `get-toml-config-source-configuration` retrieves the current configuration for the TOML Config Source

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target where the configuration source resides. Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster.

Example

Example 1 Retrieving the toml Config Source configurations.

This example gets the configuration of the toml config source.

```
asadmin get-toml-config-source-configuration
Enabled  Path                              Depth
true     /Users/payara/workspace/test.TOML  5
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- set-toml-config-source-configuration

Get

Gets the values of configurable or monitorable attributes.

Synopsis

```
asadmin [asadmin-options] get [--help]
[--monitor={true|false}]
[--aggregatedataonly={true|false}]
(dotted-attribute--name)+
```

Description

The `get` subcommand uses dotted names to get the names and values of configurable or monitorable attributes for Payara Server elements.

You can use the list subcommand to display the dotted names that represent individual server components and subsystems. For example, a dotted name might be `server.applications.web-module`. Attributes from the monitoring hierarchy are read-only, but configuration attributes can be modified using the set subcommand. For more detailed information on dotted names, see the dotted-names help page.

NOTE	Characters that have special meaning to the shell or command interpreter, such as * (asterisk), should be quoted or escaped as appropriate to the shell, for example, by enclosing the argument in

> quotes. In multimode, quotes are needed only for arguments that include spaces, quotes, or backslash.

The following list shows common usage of the `get` subcommand with the * (asterisk):

`get *` or **`get *.`**: **Gets all values on all dotted name prefixes.** ***`get domain`** or `get domain*.`: Gets all values on the dotted names that begin with `domain`. **`get *config`**..: **Gets all values on the dotted names that match `*config`.**. ***`get domain.j2ee-applications.`**.ejb-module..: **Gets all values on all EJB modules of all applications.** ***`get *web-modules.`**.: Gets all values on all web modules whether in an application or standalone. **`get *.`**..*: Gets all values on all dotted names that have four parts.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--monitor`**: **`-m`**: Defaults to `false`. If set to false, the configurable attribute values are returned. If set to `true`, the monitorable attribute values are returned. **`--aggregatedataonly`**: **`-c`**: Aggregates monitoring data for all Payara Server instances in a cluster. The default value is `false`.

Operands

dotted-attribute-name: Identifies the attribute name in the dotted notation. At least one dotted name attribute is required. The dotted notation is the syntax used to access attributes of configurable entities.

Examples

Example 1 Getting the Attributes of a Configurable Element

This example gets the attributes of `listener.http-listener-1`.

```
asadmin> get server.http-service.http-listener.http-listener-1.*
server.http-service.http-listener.http-listener-1.acceptor-threads = 1
server.http-service.http-listener.http-listener-1.address = 0.0.0.0
server.http-service.http-listener.http-listener-1.blocking-enabled = false
server.http-service.http-listener.http-listener-1.default-virtual-server = server
server.http-service.http-listener.http-listener-1.enabled = true
```

```
server.http-service.http-listener.http-listener-1.external-port =
server.http-service.http-listener.http-listener-1.family = inet
server.http-service.http-listener.http-listener-1.id = http-listener-1
server.http-service.http-listener.http-listener-1.port = 8080
server.http-service.http-listener.http-listener-1.redirect-port =
server.http-service.http-listener.http-listener-1.security-enabled = false
server.http-service.http-listener.http-listener-1.server-name =
Command get executed successfully.
```

Example 2 Getting Monitorable Objects

This example gets the configuration attributes for setting the monitoring level and shows whether they are enabled (LOW or HIGH) or disabled (OFF). The `jvm` component is enabled for monitoring.

```
asadmin> get server.monitoring-service.module-monitoring-levels.*
server.monitoring-service.module-monitoring-levels.connector-connection-pool=OFF
server.monitoring-service.module-monitoring-levels.connector-service=OFF
server.monitoring-service.module-monitoring-levels.d-trace=OFF
server.monitoring-service.module-monitoring-levels.ejb-container=OFF
server.monitoring-service.module-monitoring-levels.http-service=OFF
server.monitoring-service.module-monitoring-levels.jdbc-connection-pool=OFF
server.monitoring-service.module-monitoring-levels.jms-service=OFF
server.monitoring-service.module-monitoring-levels.jvm=HIGH
server.monitoring-service.module-monitoring-levels.orb=OFF
server.monitoring-service.module-monitoring-levels.thread-pool=OFF
server.monitoring-service.module-monitoring-levels.transaction-service=OFF
server.monitoring-service.module-monitoring-levels.web-container=OFF
Command get executed successfully.
```

Example 3 Getting Attributes and Values for a Monitorable Object

This example gets all attributes and values of the `jvm` monitorable object.

```
asadmin> get --monitor server.jvm.*
server.jvm.HeapSize_Current = 45490176
server.jvm.HeapSize_Description = Describes JvmHeapSize
server.jvm.HeapSize_HighWaterMark = 45490176
server.jvm.HeapSize_LastSampleTime = 1063217002433
server.jvm.HeapSize_LowWaterMark = 0
server.jvm.HeapSize_LowerBound = 0
server.jvm.HeapSize_Name = JvmHeapSize
server.jvm.HeapSize_StartTime = 1063238840055
server.jvm.HeapSize_Unit = bytes
server.jvm.HeapSize_UpperBound = 531628032
server.jvm.UpTime_Count = 1063238840100
server.jvm.UpTime_Description = Describes JvmUpTime
server.jvm.UpTime_LastSampleTime = 1-63238840070
server.jvm.UpTime_Name = JvmUpTime
server.jvm.UpTime_StartTime = 1063217002430
server.jvm.UpTime_Unit = milliseconds
```

```
Command get executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- list,
- set
- dotted-names
- "Payara Server General Administration section"

Import-Sync-Bundle

Imports the configuration data of a clustered instance or standalone instance from an archive file.

Synopsis

```
asadmin [asadmin-options] import-sync-bundle [--help]
--instance instance-name
[--nodedir node-dir] [--node node-name]
file-name
```

Description

The `import-sync-bundle` subcommand imports the configuration data of a clustered instance or standalone instance from an archive file that was created by the export-sync-bundle subcommand.

You must run this subcommand on the host where the instance resides. To contact the domain administration server (DAS), this subcommand requires the name of the host where the DAS is running. If a non-default port is used for administration, this subcommand also requires the port number. You must provide this information through the `--host` option and the`--port` option of the asadmin-1m utility.

Importing an instance's configuration data transfers the data to a host for an instance

without the need for the instance to be able to communicate with the DAS. Importing an instance's configuration data is typically required for the following reasons:

- To reestablish the instance after an upgrade
- To synchronize the instance manually with the domain administration server (DAS) when the instance cannot contact the DAS

The subcommand imports an instance's configuration data by performing the following operations:

- Creating or updating the instance's files and directories
- Attempting to register the instance with the DAS

If the attempt to register the instance with the DAS fails, the subcommand does not fail. Instead, the subcommand displays a warning that the attempt failed. The warning contains the command to run to register the instance with the DAS.

The `import-sync-bundle` subcommand does not contact the DAS to determine the node on which the instance resides. If the node is not specified as an option of the subcommand, the subcommand determines the node from the DAS configuration in the archive file.

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--instance`**: The instance for which configuration data is being imported. The instance must already exist in the DAS configuration. The archive file from which the data is being imported must contain data for the specified instance. **`--nodedir`**: The directory that contains the instance's node directory. The instance's files are stored in the instance's node directory. The default is as-install`/nodes`. **`--node`**: The node on which the instance resides. If this option is omitted, the subcommand determines the node from the DAS configuration in the archive file.

Operands

file-name: The name of the file, including the path, that contains the archive file to import. This operand is required.

Examples

Example 1 Importing Configuration Data for a Clustered Instance

This example imports the configuration for the clustered instance `ymli2` on the node `sj02` from the archive file `/export/payara7/glassfish/domains/domain1/sync/ymlcluster-sync-bundle.zip`.

The command is run on the host `sj02`, which is the host that the node `sj02` represents. The DAS is running on the host `sr04` and uses the default HTTP port for administration.

```
sj02# asadmin --host sr04 import-sync-bundle --node sj02 --instance ymli2
/export/payara7/glassfish/domains/domain1/sync/ymlcluster-sync-bundle.zip
Command import-sync-bundle executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- export-sync-bundle

Install-Node-Ssh

Installs Payara Server software on specified SSH-enabled hosts.

Synopsis

```
asadmin [asadmin-options] install-node-ssh [--help]
[--archive archive]
[--create={false|true}] [--save[={false|true}]
[--installdir as-install-parent]
[--sshport ssh-port] [--sshuser ssh-user]
[--sshkeyfile ssh-keyfile]
[--force={false|true}]
host-list
```

Description

The `install-node-ssh` subcommand installs Payara Server software on the hosts that are specified as the operand of the subcommand. This subcommand requires secure shell (SSH) to be configured on the host where the subcommand is run and on each host where the Payara Server software is being installed.

NOTE | This subcommand is equivalent to the install-node subcommand.

If necessary, the subcommand creates a ZIP archive of the Payara Server software from the installation where this subcommand is run. The archive does not contain the `domains` directory or the `nodes` directory. These directories are synchronized from the domain administration server (DAS) when instances on nodes that represent the hosts are created and started. The subcommand does not delete the archive after installing the Payara Server software from the archive on the specified hosts.

If multiple hosts are specified, the configuration of the following items is the same on all hosts:

- Base installation directory
- SSH port
- SSH user
- SSH key file

If the SSH key file does not exist on a host where the Payara Server software is to be installed, the subcommand runs interactively and prompts for a password. To enable the subcommand to run noninteractively, the following conditions must be met:

- The `--interactive` option of the asadmin-1m utility must be `false`.
- The `--passwordfile` option of the `asadmin` utility must specify a password file.
- The password file must contain the `AS_ADMIN_SSHPASSWORD` entry.

The subcommand does not modify the configuration of the DAS.

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--archive`**: The absolute path to the archive file of the Payara Server software that is to be installed. If no archive file is specified, the subcommand creates an archive from the installation of Payara Server software from which the subcommand is run. This archive is created in the home directory of the user that is running the command. **`--create`**: Specifies whether the subcommand should create an archive file of the Payara Server software to install.

`false`;; No archive file is created. The subcommand installs the software from the existing archive file that the `--archive` option specifies (default). `true`;; The subcommand creates an archive file from the installation of Payara Server software from which the subcommand is run. **`--save`**: Specifies whether the archive file from which the software is installed is saved after installation.

`false`;; The archive file is not saved. The subcommand deletes the file after installing the software (default). `true`;; The archive file is saved. **`--installdir`**: The absolute path to the parent of the base installation directory where the Payara Server software is to be installed on each host, for example, `/export/payara7/`. If the directory does not exist, the subcommand creates the directory.

The user that is running this subcommand must have write access to the specified directory. Otherwise, an error occurs.

To overwrite an existing an installation of the Payara Server software, set the `--force` option to `true`. If the directory already contains an installation and the `--force` option is `false`, an error occurs.

The default is the parent of the base installation directory of the Payara Server software on the host where this subcommand is run. **`--sshport`**: The port to use for SSH connections to the host where the Payara Server software is to be installed. The default is 22. **`--sshuser`**: The user on the host where the Payara Server software is to be installed that is to run the process for connecting through SSH to the host. The default is the user that is running this subcommand. To ensure that the DAS can read this

user's SSH private key file, specify the user that is running the DAS process.
`--sshkeyfile`: The absolute path to the SSH private key file for user that the `--sshuser` option specifies. This file is used for authentication to the `sshd` daemon on the host.
The user that is running this subcommand must be able to reach the path to the key file and read the key file.
The default is a key file in the user's `.ssh` directory. If multiple key files are found, the subcommand uses the following order of preference:
. `id_rsa` . `id_dsa` . `identity` `--force`: Specifies whether the subcommand overwrites an existing installation of the Payara Server software in the directory that the `--installdir` option specifies. Possible values are as follows:
`false`;; The existing installation is not overwritten (default). `true`;; The existing installation is overwritten.

Operands

host-list: A space-separated list of the names of the hosts where the Payara Server software is to be installed.

Examples

Example 1 Installing Payara Server Software at the Default Location

This example installs Payara Server software on the hosts `sj03.example.com` and `sj04.example.com` at the default location.

```
asadmin> install-node-ssh sj03.example.com sj04.example.com
Created installation zip /home/gfuser/glassfish2339538623689073993.zip
Successfully connected to gfuser@sj03.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Copying /home/gfuser/glassfish2339538623689073993.zip (81395008 bytes) to
sj03.example.com:/export/payara7
Installing glassfish2339538623689073993.zip into sj03.example.com:/export/payara7
Removing sj03.example.com:/export/payara7/glassfish2339538623689073993.zip
Fixing file permissions of all files under sj03.example.com:/export/payara7/bin
Successfully connected to gfuser@sj04.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Copying /home/gfuser/glassfish2339538623689073993.zip (81395008 bytes) to
sj04.example.com:/export/payara7
Installing glassfish2339538623689073993.zip into sj04.example.com:/export/payara7
Removing sj04.example.com:/export/payara7/glassfish2339538623689073993.zip
Fixing file permissions of all files under sj04.example.com:/export/payara7/bin
```

```
Command install-node-ssh executed successfully
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- install-node,
- uninstall-node,
- uninstall-node-ssh

Install-Node

Installs Payara Server software on specified SSH-enabled hosts

Synopsis

```
asadmin [asadmin-options] install-node [--help]
[--archive archive]
[--create={false|true}] [--save[={false|true}]
[--installdir as-install-parent]
[--sshport ssh-port] [--sshuser ssh-user]
[--sshkeyfile ssh-keyfile]
[--force={false|true}]
host-list
```

Description

The `install-node` subcommand installs Payara Server software on the hosts that are specified as the operand of the subcommand. This subcommand requires secure shell (SSH) to be configured on the host where the subcommand is run and on each host where the Payara Server software is being installed.

NOTE	This subcommand is equivalent to the install-node-ssh subcommand.

If necessary, the subcommand creates a ZIP archive of the Payara Server software from the installation where this subcommand is run. The archive does not contain the `domains` directory or the`nodes` directory. These directories are synchronized from the domain administration server (DAS) when instances on nodes that represent the hosts

are created and started. The subcommand does not delete the archive after installing the Payara Server software from the archive on the specified hosts.

If multiple hosts are specified, the configuration of the following items is the same on all hosts:

- Base installation directory
- SSH port
- SSH user
- SSH key file

If the SSH key file does not exist on a host where the Payara Server software is to be installed, the subcommand runs interactively and prompts for a password. To enable the subcommand to run non-interactively, the following conditions must be met:

- The `--interactive` option of the asadmin-1m utility must be `false`.
- The `--passwordfile` option of the `asadmin` utility must specify a password file.
- The password file must contain the `AS_ADMIN_SSHPASSWORD` entry.

The subcommand does not modify the configuration of the DAS.

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--archive`**: The absolute path to the archive file of the Payara Server software that is to be installed. If no archive file is specified, the subcommand creates an archive from the installation of Payara Server software from which the subcommand is run. This archive is created in the home directory of the user that is running the command. **`--create`**: Specifies whether the subcommand should create an archive file of the Payara Server software to install.
`false`;; No archive file is created. The subcommand installs the software from the

existing archive file that the `--archive` option specifies (default). `true`;; The subcommand creates an archive file from the installation of Payara Server software from which the subcommand is run. **`--save`**: Specifies whether the archive file from which the software is installed is saved after installation.
`false`;; The archive file is not saved. The subcommand deletes the file after installing the software (default). `true`;; The archive file is saved. **`--installdir`**: The absolute path to the parent of the base installation directory where the Payara Server software is to be installed on each host, for example, `/export/payara7/`. If the directory does not exist, the subcommand creates the directory.
The user that is running this subcommand must have write access to the specified directory. Otherwise, an error occurs.
To overwrite an existing an installation of the Payara Server software, set the `--force` option to `true`. If the directory already contains an installation and the `--force` option is `false`, an error occurs.
The default is the parent of the base installation directory of the Payara Server software on the host where this subcommand is run. **`--sshport`**: The port to use for SSH connections to the host where the Payara Server software is to be installed. The default is 22. **`--sshuser`**: The user on the host where the Payara Server software is to be installed that is to run the process for connecting through SSH to the host. The default is the user that is running this subcommand. To ensure that the DAS can read this user's SSH private key file, specify the user that is running the DAS process.
`--sshkeyfile`: The absolute path to the SSH private key file for user that the `--sshuser` option specifies. This file is used for authentication to the `sshd` daemon on the host.
The user that is running this subcommand must be able to reach the path to the key file and read the key file.
The default is a key file in the user's `.ssh` directory. If multiple key files are found, the subcommand uses the following order of preference:
. `id_rsa` . `id_dsa` . `identity` **`--force`**: Specifies whether the subcommand overwrites an existing installation of the Payara Server software in the directory that the `--installdir` option specifies. Possible values are as follows:

`false`;; The existing installation is not overwritten (default). `true`;; The existing installation is overwritten.

Operands

host-list: A space-separated list of the names of the hosts where the Payara Server software is to be installed.

Examples

Example 1 Installing Payara Server Software at the Default Location

This example installs Payara Server software on the hosts `sj03.example.com` and `sj04.example.com` at the default location.

```
asadmin> install-node sj03.example.com sj04.example.com
Created installation zip /home/gfuser/glassfish2339538623689073993.zip
Successfully connected to gfuser@sj03.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Copying /home/gfuser/glassfish2339538623689073993.zip (81395008 bytes) to
sj03.example.com:/export/payara7
Installing glassfish2339538623689073993.zip into sj03.example.com:/export/payara7
Removing sj03.example.com:/export/payara7/glassfish2339538623689073993.zip
Fixing file permissions of all files under sj03.example.com:/export/payara7/bin
Successfully connected to gfuser@sj04.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Copying /home/gfuser/glassfish2339538623689073993.zip (81395008 bytes) to
sj04.example.com:/export/payara7
Installing glassfish2339538623689073993.zip into sj04.example.com:/export/payara7
Removing sj04.example.com:/export/payara7/glassfish2339538623689073993.zip
Fixing file permissions of all files under sj04.example.com:/export/payara7/bin
Command install-node executed successfully
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- install-node-ssh,
- uninstall-node,
- uninstall-node-ssh

Jms-Ping

Checks if the JMS service is up and running.

Synopsis

```
asadmin [asadmin-options] jms-ping [--help]
[-- target target]
```

Description

The `jms-ping` subcommand checks if the Java Message Service (JMS) service (also known as the JMS provider) is up and running. When you start the Payara Server, the JMS service starts by default.

The `jms-ping` subcommand pings only the default JMS host within the JMS service. It displays an error message when it is unable to ping a built-in JMS service.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which the operation is to be performed. Valid values are as follows:
`server`;; Pings the JMS service for the default server instance. This is the default value configuration-name;; Pings the JMS service for all clusters using the specified configuration. cluster-name;; Pings the JMS service for the specified cluster. instance-name;; Pings the JMS service for the specified server instance.

Examples

Example 1 Verifying that the JMS service is running

The following subcommand checks to see if the JMS service is running on the default server.

```
asadmin> jms-ping
JMS-ping command executed successfully
```

```
Connector resoure test_jms_adapter created.
Command jms-ping executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jms-host,
- delete-jms-host,
- list-jms-hosts

List-Admin-Objects

Gets all the administered objects.

Synopsis

```
asadmin [asadmin-options] list-admin-objects [--help]
[target]
```

Description

The `list-admin-objects` subcommand lists all the administered objects.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: The target for which administered objects are to be listed. Valid values are as follows:

`server`;; Lists the administered objects on the default server instance. This is the default value. configuration-name;; Lists the administered objects in the specified

configuration. cluster-name;; Lists the administered objects on all server instances in the specified cluster. instance-name;; Lists the administered objects on a specified server instance.

Examples

Example 1 Listing Administered Objects

This example lists all the administered objects.

```
asadmin> list-admin-objects
jms/samplequeue
jms/anotherqueue
Command list-admin-objects executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-admin-object,
- delete-admin-object

List-Application-Refs

Lists the existing application references.

Synopsis

```
asadmin [asadmin-options] list-application-refs [--help]
[--long={false|true}] [target]
```

Description

The `list-application-refs` subcommand lists all application references in a cluster or an unclustered server instance. This effectively lists all the modules deployed on the specified target (for example, J2EE applications, Web modules, and enterprise bean modules).

If multiple versions of a module or application are deployed, this subcommand lists all versions. To list which version is enabled, set the `--long` option to `true`. For more information about module and application versions, see module-and-application-versions in Payara Server Application Deployment Guide.

The target instance or instances making up the cluster need not be running or available for this subcommand to succeed.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--long`**: If `true`, displays whether each module or application listed is enabled. The default is `false`.

Operands

target: The target for which you are listing the application references. Valid values are * `server`- Specifies the default server instance as the target. `server` is the name of the default server instance and is the default value. * cluster_name- Specifies a certain cluster as the target. * instance_name- Specifies a certain server instance as the target.

Examples

Example 1 Listing Application References

The following example lists the application references for the unclustered server instance `NewServer`.

```
asadmin> list-application-refs NewServer
ClientSessionMDBApp
MEjbApp
__ejb_container_timer_app
Command list-application-refs executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-application-ref,
- delete-application-ref

List-Applications

Lists deployed applications.

Synopsis

```
asadmin [asadmin-options] list-applications [--help]
[--long={false|true}] [--resources] [--subcomponents]
[--type type] [target]
```

Description

The `list-applications` subcommand lists deployed Jakarta EE applications and the type of each application that is listed.

If the `--type` option is not specified, all applications are listed. If the type option is specified, you must specify a type. The possible types are listed in the Options section of this help page.

If multiple versions of a module or application are deployed, this subcommand lists all versions. To list which version is enabled, set the `--long` option to `true`. For more information about module and application versions, see module-and-application-versions in Payara Server Application Deployment Guide.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--long`**: If `true`, displays whether each module or application listed is enabled. The default is `false`. **`--resources`**: Lists the application-scoped resources for each application. If the `--subcomponents` option is also used, lists the application-

scoped resources for each component within the application. `--subcomponents`: Lists the subcomponents of each application. The subcomponents listed depend on the application type. For example, for a Jakarta EE application (EAR file), modules are listed. For a web application, servlets and JSP pages are listed. For an EJB module, EJB subcomponents are listed. `--type`: Specifies the type of the applications that are to be listed. The options are as follows:

* `application` * `appclient` * `connector` * `ejb` * `web` * `webservice`

If no type is specified, all applications are listed.

Operands

`--target`: This is the name of the target upon which the subcommand operates. The valid values are as follows:

`server`;; Lists the applications for the default server instance `server` and is the default value. `domain`;; Lists the applications for the domain. cluster_name;; Lists the applications for the cluster. instance_name;; Lists the applications for a particular stand-alone server instance.

Examples

Example 1 Listing the Web Applications

```
asadmin> list-applications --type web
hellojsp <web>
Command list-applications executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-sub-components,
- show-component-status

List-Auth-Realms

Lists the authentication realms.

Synopsis

```
asadmin [asadmin-options] list-auth-realms [--help]
[target]
```

Description

The `list-auth-realms` subcommand lists the authentication realms. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: The name of the target for which you want to list the authentication realms. `server`;; Lists the realms for the default server instance `server` and is the default value. configuration_name;; Lists the realms for the named configuration. cluster_name;; Lists the realms for every server instance in the cluster. instance_name;; Lists the realms for a particular server instance.

Examples

Example 1 Listing authentication realms

```
asadmin> list-auth-realms
file
ldap
certificate
db
Command list-auth-realms executed successfully
```

Where file, ldap, certificate, and db are the available authentication realms.

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-auth-realm,
- delete-auth-realm

List-Backups

Lists all backups.

Synopsis

```
asadmin [asadmin-options] list-backups [--help]
[--long[={false|true}]]
[--domaindir domain-root-dir]
[--backupdir backup-directory]
[--backupconfig backup-config-name]
[domain-name]
```

Description

The `list-backups` subcommand displays information about domain backups.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--long`**: **`-l`**: Displays detailed information about each backup.
The default value is `false`. **`--domaindir`**: Specifies the domain root directory, the parent directory of the domain upon which the command will operate.
The default value is `as-install/domains`. **`--backupdir`**: Specifies the directory under which backup files are stored.
The default value is `as-install/domains/domain-dir/backups`. If the domain is not in the default location, the location is `domain-dir/backups`. **`--backupconfig`**: Restricts the listing of backup files in the backup directory to those for the specified domain backup configuration.

Operands

domain-name: Specifies the domain for which backups are listed.

This operand is optional if only one domain exists in the Payara Server installation.

Examples

Example 1 Listing Domain Backups

This example provides detailed information about backups in the default domain.

```
asadmin> list-backups --long
Description               : domain1 backup created on 2023_12_01 by user adminuser
GlassFish Version         : Payara Server 7.2.0 #badassfish
Backup User               : adminuser
Backup Date               : Wed Dec 01 09:22:45 PST 2023
Domain Name               : domain1
Backup Type               : full
Backup Config Name        :
Backup Filename (origin)  :
/payara/glassfish/domains/domain1/backups/domain1_2023_12_01_v00001.zip
Domain Directory          : /payara/glassfish/domains/domain1

Description               : domain1 backup created on 2023_12_16 by user adminuser
GlassFish Version         : Payara Server 7.2.0 #badassfish
Backup User               : adminuser
Backup Date               : Thu Dec 16 09:23:48 PST 2023
Domain Name               : domain1
Backup Type               : full
Backup Config Name        :
Backup Filename (origin)  :
/payara/glassfish/domains/domain1/backups/domain1_2023_12_16_v00001.zip
Domain Directory          : /payara/glassfish/domains/domain1

Command list-backups executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- backup-domain,
- restore-domain

List-Batch-Job-Executions

Lists batch job executions and execution details.

Synopsis

```
asadmin [asadmin-options] list-batch-job-executions [--help]
[--target target]
[--executionid execution-id]
[--long={false|true}]
[--output output]
[--header={false|true}]
[instance_ID]
```

Description

The `list-batch-job-executions` subcommand lists batch job executions and execution details.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--target`: Specifies the target for which to list batch job executions and execution details. Valid values are as follows:
`server`;; Lists executions for the default server instance `server` and is the default value. cluster-name;; Lists executions for every server instance in the cluster. instance-name;; Lists executions for a particular server instance. `--executionid`: `-x`: Specifies the execution ID of a specific batch job execution. `--long`: `-l`: Displays detailed information about batch job executions. The default value is `false`. `--output`: `-o`: Displays specific details about batch job executions. Use a comma-separated list to specify the details to display and their order. The values are case-insensitive. A subset of all possible headings is displayed by default.
Possible values are as follows:
`jobname`;; Displays the name of the job. `executionid`;; Displays the ID assigned to the execution of the batch job. A new execution is created the first time a job is started and every time the existing execution is restarted. `starttime`;; Displays the start time of the execution. `endtime`;; Displays the finish time of the execution. `batchstatus`;; Displays the status of the execution as set by the batch runtime. `exitstatus`;; Displays the status of the execution as set by the Job XML for the job or by the batch application. By default, the exit status and the batch status are the same unless the exit status is explicitly overridden. `jobparameters`;; Displays the properties passed to the

batch runtime for the batch job execution, listed as name/value pairs. `stepcount`;; Displays the number of steps in the batch job execution. **`--header`**: **`-h`**: Specifies whether column headings are displayed when the `--long` option is used. The default value is `true`. To suppress the headings, set the `--header` option to `false`.

Operands

instance_id: The ID of the job instance for which to list execution details.

Examples

Example 1 Listing Batch Job Executions

This example lists batch job executions for the default server instance and displays specific details.

```
asadmin> list-batch-job-executions -o=jobname,executionid,batchstatus,exitstatus
JOBNAME  EXECUTIONID  BATCHSTATUS  EXITSTATUS
payroll  9            COMPLETED    COMPLETED
bonus    6            FAILED       FAILED
Command list-batch-job-executions executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-batch-jobs,
- list-batch-job-steps,
- list-batch-runtime-configuration,
- set-batch-runtime-configuration

List-Batch-Job-Steps

Lists steps for a specific batch job execution.

Synopsis

```
asadmin [asadmin-options] list-batch-job-steps [--help]
```

```
[--long={false|true}]
[--target target]
[--output output]
[--header={false|true}]
execution_id
```

Description

The `list-batch-job-steps` subcommand lists steps for a specific batch job execution.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which to list batch job steps. Valid values are as follows:

`server`;; Lists steps for the default server instance `server` and is the default value. cluster-name;; Lists steps for every server instance in the cluster. instance-name;; Lists steps for a particular server instance. **`--long`**: **`-l`**: Displays detailed information about batch job steps. The default value is `false`. **`--output`**: **`-o`**: Displays specific details about batch job steps. Use a comma-separated list to specify the details to display and their order. The values are case-insensitive. A subset of all possible headings is displayed by default.

Possible values are as follows:

`stepname`;; Displays the name of the step. `stepid`;; Displays the step ID. `starttime`;; Displays the start time of the step. `endtime`;; Displays the finish time of the step. `batchstatus`;; Displays the status of the step as set by the batch runtime. `exitstatus`;; Displays the status of the step as set by the Job XML for the job or by the batch application. By default, the exit status and the batch status are the same unless the exit status is explicitly overridden. `stepmetrics`;; Displays metrics for the step. `value`;; Displays a value for each step metric. The value represents the number of items read, written, committed, and so on. **`--header`**: **`-h`**: Specifies whether column headings are displayed when the `--long` option is used. The default value is `true`. To suppress the headings, set the `--header` option to `false`.

Operands

execution_id: The ID of the execution for which to list batch job steps and details.

Examples

Example 1 Listing Batch Job Steps

This example lists batch job steps and specific step details for a job execution with the execution ID of `7`. The target is the default server instance.

Some lines of output are omitted from this example for readability.

```
asadmin> list-batch-job-steps o=stepname,stepid,batchstatus,stepmetrics 7
STEPNAME   STEPID   BATCHSTATUS   STEPMETRICS
prepare    7        COMPLETED     METRICNAME          VALUE
                                  READ_COUNT          8
                                  WRITE_COUNT         8
                                  PROCESS_SKIP_COUNT  0
process    8        COMPLETED     METRICNAME          VALUE
                                  READ_COUNT          8
                                  WRITE_COUNT         8
                                  PROCESS_SKIP_COUNT  0
...
Command list-batch-job-steps executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-batch-jobs,
- list-batch-job-executions,
- list-batch-runtime-configuration,
- set-batch-runtime-configuration

List-Batch-Jobs

Lists batch jobs details.

Synopsis

```
asadmin [asadmin-options] list-batch-jobs [--help]
[--target target]
[--long={false|true}]
[--output output]
[--limit] limit
[--offset] offset
[--header={false|true}]
[job_name]
```

Description

The `list-batch-jobs` subcommand lists batch jobs and job details.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which to list batch jobs and job details. Valid values are as follows:
`server`;; Lists batch jobs for the default server instance `server` and is the default value. cluster-name;; Lists batch jobs for every server instance in the cluster. instance-name;; Lists batch jobs for a particular server instance. **`--long`**: **`-l`**: Displays detailed information about batch jobs. The default value is `false`. **`--limit`**: Sets the number of rows of batch jobs to display **`--offset`**: Sets the number of rows of batch jobs to skip **`--output`**: **`-o`**: Displays specific details about batch jobs. Use a comma-separated list to specify the details to display and their order. The values are case-insensitive. The `jobname` and `instancecount` column headings are displayed by default.
Possible values are as follows:
`jobname`;; Displays the name of the job. `appname`;; Displays the name of the application. `instancecount`;; Displays the number of job instances. `instanceid`;; Displays the ID assigned to the job instance. `executionid`;; Displays the ID assigned to the execution of the batch job. A new execution is created the first time a job is started and every time the existing execution is restarted. `batchstatus`;; Displays the status of the job as set by the batch runtime. `starttime`;; Displays the start time of the job. `endtime`;; Displays the finish time of the job. `exitstatus`;; Displays the status of the job as set by the Job XML for the job or by the batch application. By default, the exit

status and the batch status are the same unless the exit status is explicitly overridden.
`--header`: `-h`: Specifies whether column headings are displayed when the `--long` option is used. The default value is `true`. To suppress the headings, set the `--header` option to `false`.

Operands

job_name: The name of the job for which to list details.

Examples

Example 1 Listing Batch Jobs

This example lists batch jobs for the default server instance.

```
asadmin> list-batch-jobs
JOBNAME  INSTANCECOUNT
payroll  9
bonus    6
Command list-batch-jobs executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-batch-job-executions,
- list-batch-job-steps,
- list-batch-runtime-configuration,
- set-batch-runtime-configuration

List-Batch-Runtime-Configuration

Displays the configuration of the batch runtime.

Synopsis

```
asadmin [asadmin-options] list-batch-runtime-configuration [--help]
[--target target]
[--output output]
```

```
[--header={false|true}]
```

Description

The `list-batch-runtime-configuration` subcommand displays the configuration of the batch runtime. Batch runtime configuration data is stored in the `config` element in `domain.xml`.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which to list the batch runtime configuration. Valid values are as follows:
`server`;; Lists the batch runtime configuration for the default server instance `server` and is the default value. cluster-name;; Lists the batch runtime configuration for every server instance in the cluster. instance-name;; Lists the batch runtime configuration for a particular server instance. **`--output`**: **`-o`**: Displays specific details about the batch runtime configuration. Use a comma-separated list to specify the details to display and their order. The values are case-insensitive. The `datasourcelookupname` and `executorservicelookupname` column headings are displayed by default.
Possible values are as follows:
`datasourcelookupname`;; The JNDI lookup name of the data source used to store job information. By default, the batch runtime uses the default data source `jdbc/TimerPool`. `executorservicelookupname`;; The JNDI lookup name of the managed executor service used to provide threads to jobs. By default, the batch runtime uses the default managed executor service `concurrent/defaultManagedExecutorService`. **`--header`**: **`-h`**: Specifies whether column headings are displayed when the `--long` option is used. The default value is `true`. To suppress the headings, set the `--header` option to `false`.

Examples

Example 1 Listing Batch Runtime Configuration

The following example lists the configuration of the batch runtime for the default server instance.

```
asadmin> list-batch-runtime-configuration
DATASOURCELOOKUPNAME     EXECUTORSERVICELOOKUPNAME
jdbc/_default            concurrent/__defaultManagedExecutorService
Command list-batch-runtime-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-batch-runtime-configuration,
- list-batch-jobs,
- list-batch-job-executions,
- list-batch-job-steps

List-Cache-Keys

Lists all keys in a Hazelcast cache.

Synopsis

```
asadmin [asadmin-options] list-cache-keys
[--help]
[--target target]
[--name name]
```

Description

This command can be used to list all keys in a Hazelcast (Data Grid) cache.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are executing this operation. Valid values are:
`server`;; Applies to the default server instance. This is the default value.
instance_name;; Applies to a specified server instance. **`--name`**: Name of the cache to

query.

+ TIP: If omitted, this command will be executed for all available caches in the corresponding target.

Examples

Example 1 Lists all keys in all available caches.

```
asadmin> list-cache-keys
{
Cache payara-instance-map
{5d0928cf-fd3d-404f-a6c6-985293f0295f,
c7d90acc-43a3-492d-8c9c-8cd9b950fb37,
5e984e01-0c6a-4b34-b608-56b79285c36b,
9ba96a9f-d7ff-4c79-acef-8dd8091f40a6,
Cache payara.instance.store
{5d0928cf-fd3d-404f-a6c6-985293f0295f,
5e984e01-0c6a-4b34-b608-56b79285c36b,
Cache payara.micro.applications.store
{__admingui,
}

Command list-cache-keys executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Domain Data Grid",
- "`list-hazelcast-cluster-members`",
- "`list-hazelcast-members`",
- "`restart-hazelcast`",
- "`list-caches`",
- "`clear-cache`",
- "`set-hazelcast-configuration`",
- "`get-hazelcast-configuration`",

- "`generate-encryption-key`"

List-Caches

Lists all configured Hazelcast distributed caches.

Synopsis

```
asadmin [asadmin-options] list-caches
[--help]
[--target target]
```

Description

This command can be used to list the Hazelcast distributed caches configured in the corresponding target.

For each cache entry, a minimal amount of information is printed out.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are executing this operation. Valid values are:
`server`;; Applies to the default server instance. This is the default value.
instance_name;; Applies to a specified server instance.

Examples

Example 1 Lists the Hazelcast distributed caches in the domain.

```
asadmin> list-caches
{
HZMap payara-instance-map hz:impl:mapService 4 entries
HZMap payara.instance.store hz:impl:mapService 2 entries
HZMap payara.micro.applications.store hz:impl:mapService 1 entries
}

Command list-caches executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Domain Data Grid",
- "`list-hazelcast-cluster-members`",
- "`list-hazelcast-members`",
- "`restart-hazelcast`",
- "`list-cache-keys`",
- "`clear-cache`",
- "`set-hazelcast-configuration`",
- "`get-hazelcast-configuration`",
- "`generate-encryption-key`"

List-Clusters

Lists existing clusters in a domain.

Synopsis

```
asadmin [asadmin-options] list-clusters [--help]
[target]
```

Description

The `list-clusters` subcommand lists existing clusters in a domain. The list can be filtered by cluster, instance, node, or configuration. For each cluster that is listed, the subcommand indicates whether the cluster is running.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: Filters the list of clusters by specifying the target for which the clusters are to be listed. Valid values are as follows:
`domain`;; Lists all clusters in the domain (default). cluster-name;; Lists only the specified cluster. instance-name;; Lists the cluster of which the specified instance is a member. node-name;; Lists the clusters that contain an instance that resides on the specified node. For example, if instance `pmdi1` in cluster `pmdc` and instance `ymli1` in cluster `ymlc` reside on node `n1`, `pmdc` and `ymlc` are listed. configuration-name;; Lists all clusters that contain instances whose configuration is defined by the named configuration.

Examples

Example 1 Listing All Clusters in a Domain

This example lists all clusters in the current domain.

```
asadmin> list-clusters
pmdclust not running
ymlclust not running

Command list-clusters executed successfully.
```

Example 2 Displaying the Status of a Cluster

This example displays status of the cluster `ymlclust`, which is not running.

```
asadmin> list-clusters ymlclust
ymlclust not running

Command list-clusters executed successfully.
```

Example 3 Listing All Clusters That Are Associated With a Node

This example lists the clusters that contain an instance that resides on the node `sj02`.

```
asadmin> list-clusters sj02
ymlclust not running

Command list-clusters executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-cluster,
- delete-cluster,
- start-cluster,
- stop-cluster,
- restart-cluster

List-Commands

Lists available commands.

Synopsis

```
asadmin [asadmin-options] list-commands [--help]
[--localonly={false|true}] [--remoteonly={false|true}]

pattern-list
```

Description

The `list-commands` subcommand lists the `asadmin` subcommands.

By default, the `list-commands` subcommand displays a list of local subcommands followed by a list of remote subcommands. You can specify that only remote subcommands or only local subcommands are listed and that only subcommands whose names contain a specified text string are listed.

This subcommand is supported in local mode and remote mode.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--localonly`**: If this option is set to true, only local commands are

listed. Default is false.

If this option is set to true, the `--remoteonly` option must be set to false. Otherwise, an error occurs. **`--remoteonly`**: If this option is set to true, only remote commands are listed. Default is false.

If this option is set to true, the `--localonly` option must be set to false. Otherwise, an error occurs.

Operands

pattern-list: A space-separated list of text strings on which to filter the list of subcommands. Only the subcommands that contain any one of the specified text strings is listed.

Examples

Example 1 Listing the Local Subcommands

This example lists only the local subcommands.

```
asadmin> list-commands --localonly=true
********** Local Commands **********
change-admin-password
change-master-password
create-domain
create-service
delete-domain
export
help
list-commands
list-domains
login
monitor
multimode
restart-domain
start-database
start-domain
stop-database
stop-domain
unset
verify-domain-xml
version
Command list-commands executed successfully.
```

Example 2 Filtering the Subcommands That Are Listed

This example lists only the subcomands whose names contain the text `configure` or

`set`.

```
asadmin> list-commands configure set
********** Local Commands **********
setup-ssh
unset

********** Remote Commands **********
configure-jms-cluster                   set-log-levels
configure-lb-weight                     set-web-context-param
configure-ldap-for-admin                set-web-env-entry
set                                     unset-web-context-param
set-log-attributes                      unset-web-env-entry

Command list-commands executed successfully.
```

Example 3 Listing All Subcommands

This example first displays a list of the local subcommands, followed by a partial list of the remote subcommands.

```
asadmin> list-commands
********** Local Commands **********
change-admin-password
change-master-password
create-domain
create-service
delete-domain
export
help
list-commands
list-domains
login
monitor
multimode
restart-domain
start-database
start-domain
stop-database
stop-domain
unset
verify-domain-xml
version
********** Remote Commands **********
__locations                             enable
add-resources                           enable-monitoring
configure-ldap-for-admin                flush-jmsdest
create-admin-object                     freeze-transaction-service
                                        generate-jvm-report
create-auth-realm                       get
create-connector-connection-pool        get-client-stubs
create-connector-resource               get-host-and-port
create-connector-security-map           jms-ping
create-connector-work-security-map      list
```

```
create-custom-resource                  list-admin-objects
create-file-user                        list-app-refs
create-http                             list-applications
create-http-listener
create-iiop-listener                    list-auth-realms
create-javamail-resource
create-jdbc-connection-pool             list-connector-connection-pools
create-jdbc-resource                    list-connector-resources
create-jms-host                         list-connector-security-maps
create-jms-resource                     list-connector-work-security-maps
create-jmsdest                          list-containers
create-jndi-resource                    list-custom-resources
create-jvm-options                      list-file-groups
                                        list-file-users
create-message-security-provider        list-http-listeners
create-network-listener                 list-iiop-listeners
create-password-alias                   list-javamail-resources
                                        list-jdbc-connection-pools
create-protocol                         list-jdbc-resources
create-resource-adapter-config          list-jms-hosts
create-resource-ref                     list-jms-resources
create-ssl                              list-jmsdest
create-system-properties                list-jndi-entries
create-threadpool                       list-jndi-resources
create-transport                        list-jvm-options
create-virtual-server
delete-admin-object                     list-logger-levels
                                        list-message-security-providers
...
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-containers,
- list-modules

List-Configs

Lists named configurations.

Synopsis

```
asadmin [asadmin-options] list-configs [--help]
[target]
```

Description

The `list-configs` subcommand lists named configurations in the configuration of the domain administration server (DAS). The list can be filtered by cluster, instance, or named configuration.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: Filters the list of configurations. Valid values are as follows: `domain`;; Lists all named configurations in the current domain. cluster-name;; Lists the named configuration that defines the configuration of instances in the specified cluster. instance-name;; Lists the named configuration that defines the configuration of the specified instance. configuration-name;; Lists the specified named configuration. Use this option to determine whether a named configuration exits.

Examples

Example 1 Listing All Named Configurations in a Domain

This example lists all named configurations in the current domain.

```
asadmin> list-configs
server-config
default-config
pmdclust-config
pmdsharedconfig
pmdcpinst-config
yml1clust-config
il1-config
il2-config

Command list-configs executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- copy-config,
- delete-config
- configuration

List-Connector-Connection-Pools

Lists the existing connector connection pools.

Synopsis

```
asadmin [asadmin-options] list-connector-connection-pools [--help]
```

Description

The `list-connector-connection-pools` subcommand list connector connection pools that have been created.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 Listing the Connector Connection Pools

This example lists the existing connector connection pools.

```
asadmin> list-connector-connection-pools
jms/qConnPool
Command list-connector-connection-pools executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-connector-connection-pool,
- delete-connector-connection-pool,
- ping-connection-pool

List-Connector-Resources

Lists all connector resources.

Synopsis

```
asadmin [asadmin-options] list-connector-resources [--help]
[target]
```

Description

The `list-connector-resources` subcommand lists all connector resources.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: The target for which the connector resources are to be listed. Valid values are as follows:

`server`;; Lists the connector resources on the default server instance. This is the default value. `domain`;; Lists the connector resources for the domain. cluster-name;; Lists the connector resources on all server instances in the specified cluster. instance-name;; Lists the connector resources on a specified server instance.

Examples

Example 1 Listing Connector Resources

This example lists all existing connector resources.

```
asadmin> list-connector-resources
jms/qConnFactory
Command list-connector-resources executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-connector-resource,
- delete-connector-resource

List-Connector-Security-Maps

Lists the security maps belonging to the specified connector connection pool.

Synopsis

```
asadmin [asadmin-options] list-connector-security-maps [--help]
[--securitymap securitymap]
[--verbose={false|true}]
pool-name
```

Description

The `list-connector-security-maps` subcommand lists the security maps belonging to the specified connector connection pool.

For this subcommand to succeed, you must have first created a connector connection pool using the `create-connector-connection-pool` subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--securitymap`**: Specifies the name of the security map contained

within the connector connection pool from which the identity and principals should be listed. With this option, `--verbose` is redundant. **`--verbose`**: If set to `true`, returns a list including the identity, principals, and security name. The default is `false`.

Operands

pool-name: Name of the connector connection pool for which you want to list security maps.

Examples

Example 1 Listing the Connector Security Maps

This example lists the existing connector security maps for the pool named `connector-Pool1`.

```
asadmin> list-connector-security-maps connector-Pool1
securityMap1
Command list-connector-security-maps executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-connector-security-map,
- delete-connector-security-map,
- update-connector-security-map

List-Connector-Work-Security-Maps

Lists the work security maps belonging to the specified resource adapter.

Synopsis

```
asadmin [asadmin-options] list-connector-work-security-maps [--help]
[--securitymap securitymap]
resource_adapter_name
```

Description

The `list-connector-work-security-maps` subcommand lists the work security maps belonging to the specified resource adapter.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--securitymap`**: Specifies the name of the security map contained within the resource adapter from which the identity and principals should be listed.

Operands

resource_adapter_name: The name of the resource adapter for which you want to list security maps.

Examples

Example 1 Listing Connector Work Security Maps

This example lists the current connector work security maps for the resource adapter named `my_resource_adapter`.

```
asadmin> list-connector-work-security-maps my_resource_adapter
workSecurityMap1: EIS principal=eis-principal-2, mapped principal=server-principal-
2
workSecurityMap1: EIS principal=eis-principal-1, mapped principal=server-principal-
1
workSecurityMap2: EIS principal=eis-principal-2, mapped principal=server-principal-
2
workSecurityMap2: EIS principal=eis-principal-1, mapped principal=server-principal-
1
Command list-connector-work-security-maps executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,

- create-connector-work-security-map,
- delete-connector-work-security-map,
- update-connector-work-security-map

List-Containers

Lists application containers.

Synopsis

```
asadmin [asadmin-options] list-containers [--help]
```

Description

The `list-containers` subcommand displays a list of application containers.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 Listing the Application Containers

This example lists the current application containers.

```
asadmin> list-containers
List all known application containers

faulttolerance
metrics
appclient
jpa
jpa
osgi
connector
grizzly
health
jwtauth
ejb
openapi
webservices
security
```

```
payara-resources_ear
payara-resources
resources_ear
resources
web
ear
cdi
cdi
config

Command list-containers executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-commands,
- list-modules

List-Context-Services

Lists context service resources.

Synopsis

```
asadmin [asadmin-options] list-context-services [--help]
[target]
```

Description

The `list-context-services` subcommand lists context service resources.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: Specifies the target for which context service resources are to be listed. Valid

targets are:

+ `server`;; Lists the resources on the default server instance. This is the default value. `domain`;; Lists the resources for the domain. cluster-name;; Lists the resources on all server instances in the specified cluster. instance-name;; Lists the resources on a specified server instance.

Examples

Example 1 Listing Context Service Resources

This example lists context service resources on the default server instance.

```
asadmin> list-context-services
concurrent/__defaultContextService
concurrent/myContextService1
concurrent/myContextService2
Command list-context-services executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-context-service,
- delete-context-service

List-Custom-Resources

Gets all custom resources.

Synopsis

```
asadmin [asadmin-options] list-custom-resources [--help]
[target]
```

Description

The `list-custom-resources` subcommand lists the custom resources.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: This operand specifies the location of the custom resources. Valid targets are: `server`;; Lists the resources on the default server instance. This is the default value `domain`;; Lists the resources in the domain. cluster_name;; Lists the resources for every server instance in the cluster. instance_name;; Lists the resources for a particular server instance.

Examples

Example 1 Listing Custom Resources

This example lists the current custom resources.

```
asadmin> list-custom-resources
sample_custom_resource01
sample_custom_resource02
Command list-custom-resources executed successfully.
```

Example 2 Using the list-custom-resources command with a target

The following example displays the usage of this command.

```
asadmin> list-custom-resources --user admin --passwordfile
passwords.txt --host plum --port 4848 target6

sample_custom_resource03
sample_custom_resource04
Command list-custom-resources executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-custom-resource,
- delete-custom-resource

List-Deployment-Groups

List all deployment groups registered in an existing server domain.

Synopsis

```
asadmin [asadmin-options] list-deployment-groups [--help]
```

Description

The `list-deployment-groups` subcommand lists all existing deployment groups registered in the server domain.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: `-?`: Displays the help text for the subcommand.

Examples

Example 1 Listing Deployment Groups

This example lists existing deployment groups created in the domain:

```
asadmin list-deployment-groups
List of Deployment Groups:
        test-dgroup1
        test-dgroup2

Command list-deployment-groups executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-deployment-group,
- delete-deployment-group,
- start-deployment-group,
- stop-deployment-group,
- restart-deployment-group

List-Domains

Lists the domains in the specified directory.

Synopsis

```
asadmin [asadmin-options] list-domains [--help]
[--domaindir domaindir]
[--long={false|true}]
[--header={false|true}]
```

Description

The `list-domains` subcommand lists the domains in the specified domains directory. If the domains directory is not specified, the domains in the default directory are listed. If there is more that one domains directory, the `--domaindir` option must be specified. The status of each domain is included.

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--domaindir`**: The directory where the domains are to be listed. If specified, the path must be accessible in the files stem. If not specified, the domains in the domain root directory are listed. The default location of the domain root directory is `as-install/domains`. **`--long`**: **`-l`**: Displays detailed information about the administration servers in the listed domains, including host names and port numbers.

The default value is `false`. **`--header`**: **`-h`**: Specifies whether a header is displayed when the `--long` option is used. The default value is `true`. To suppress the header, set the `--header` option to `false`.

Examples

Example 1 Listing Domains

This example lists the domains in the default directory.

```
asadmin> list-domains
Name: domain1 Status: Running
Name: domain2 Status: Not running
Name: domain4 Status: Running, restart required to apply configuration changes
Command list-domains executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-domain,
- delete-domain,
- start-domain,
- stop-domain

List-File-Groups

Lists file groups.

Synopsis

```
asadmin [asadmin-options] list-file-groups [--help]
[--name username] [--authrealmname auth_realm_name]
[--target target]
```

Description

The `list-file-groups` subcommand lists the file users and groups supported by the

file realm authentication. This subcommand lists available groups in the file user. If the `--name` option is not specified, all groups are listed.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--name`**: Identifies the name of the file user for whom the groups will be listed. **`--authrealmname`**: The name of the authentication realm for which to list available groups. **`--target`**: This option specifies which configurations you can list. Valid targets are:
`server`;; Lists the file groups in the current server. This is the default value. cluster_name;; Lists the file groups in a cluster. instance_name;; Lists the file groups for a particular instance.

Examples

Example 1 Listing Groups in all File Realms

This example list all file realm groups defined for the server.

```
asadmin> list-file-groups
staff
manager
Command list-file-groups executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-file-user,
- delete-file-user,
- list-file-users,

- update-file-user

List-File-Users

Lists the file users.

Synopsis

```
asadmin [asadmin-options] list-file-users [--help]
[--authrealmname auth_realm_name] [target]
```

Description

The `list-file-users` subcommand displays a list of file users supported by file realm authentication.

Options

`--help`: **`-?`**: Displays the help text for the subcommand. **`--authrealmname`**: Lists only the users in the specified authentication realm.

Operands

target: Specifies the target for which you want to list file users. The following values are valid:

`server`;; Lists the file users on the default server instance. This is the default value. configuration_name;; Lists the file users in the specified configuration. cluster_name;; Lists the file users on all server instances in the specified cluster. instance_name;; Lists the file users on a specified server instance.

Examples

Example 1 Listing Users in a Specific File Realm

The following example lists the users in the file realm named `sample_file_realm`.

```
asadmin> list-file-users --authrealmname sample_file_realm
sample_user05
sample_user08
sample_user12
Command list-file-users executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-file-user,
- delete-file-user,
- list-file-groups,
- update-file-user

List-Hazelcast-Cluster-Members

List the current members of the Data Grid visible to the domain.

Synopsis

```
asadmin [asadmin-options] list-hazelcast-cluster-members
[--help]
[--type type]
```

Description

This command can be used to list the current members of the Data Grid in a formatted table.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--type`**: Specifies the type of cluster members to list, which will be either `server` or `micro`. The default value is `both`.

Examples

Example 1 List the current members of the Data Grid visible to the domain

```
asadmin> list-hazelcast-cluster-members --type server
Instance Name  Instance Group  Instance Type  Host Name     HTTP Ports  HTTPS Ports
Admin Port  Hazelcast Port  Lite Member  Deployed Applications  Last Heartbeat
instance1      MicroShoal      INSTANCE       /172.18.64.1  28080       28181
24848       2900           false                              2024-03-26 01:23:23
```

```
server          MicroShoal      DAS             /172.18.64.1  8080        8181
4848        4900            false        __admingui             2024-03-26 01:23:20
Command list-hazelcast-cluster-members executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Domain Data Grid",
- "`list-hazelcast-members`",
- "`restart-hazelcast`",
- "`list-cache-keys`",
- "`list-caches`",
- "`clear-cache`",
- "`set-hazelcast-configuration`",
- "`get-hazelcast-configuration`",
- "`generate-encryption-key`"

List-Hazelcast-Members

List the Data Grid (Hazelcast) current members.

Synopsis

```
asadmin [asadmin-options] list-hazelcast-members
[--help]
[--target target]
```

Description

This command can be used to list all Data Grid members (including the DAS, server instances, Payara Micro instances, other nodes) that correspond to the selected target.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options,

see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are executing this operation. Valid values are:
`server`;; Applies to the default `server` instance. This is the default value.
instance_name;; Applies to a specified server instance.

Examples

Example 1 List current Data Grid members

```
asadmin> list-hazelcast-members
{ instance1-MicroShoal-/172.18.64.1:2900 server-MicroShoal-/172.18.64.1:4900-this }
Command list-hazelcast-members executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Domain Data Grid",
- "`list-hazelcast-cluster-members`",
- "`restart-hazelcast`",
- "`list-cache-keys`",
- "`list-caches`",
- "`clear-cache`",
- "`set-hazelcast-configuration`",
- "`get-hazelcast-configuration`",
- "`generate-encryption-key`"

List-Healthcheck-Services

Lists the names of all available Healthcheck services.

Synopsis

```
asadmin [asadmin-options] list-healthcheck-services
[--help]
```

Description

Prints a detailed list of all available Healthcheck services (as in, individual services that generate healthcheck events based on performance metrics) in the current server installation.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 List current Healthcheck services

```
asadmin list-healthcheck-services

Available Health Check Services:
Name                    Description
healthcheck-mpmetrics   Provides a way to monitor and log the values of metrics
exposed by MicroProfile Metrics
healthcheck-cpu         Provides ratio on cpu usage time with severity according to
defined threshold values
healthcheck-gc          Provides ratio on garbage collection count with severity
according to defined threshold values
healthcheck-heap        Provides ratio on used heap memory with severity according
to defined threshold values
healthcheck-threads     Lists hogging threads with their id when given thresholds
exceed
healthcheck-machinemem  Provides ratio on used machine memory with severity
according to defined threshold values
healthcheck-cpool       Provides ratio on connection usage for a given pool name
with severity according to defined threshold values
healthcheck-stuck       Provides thread name, id and stack trace for requests which
reach over defined threshold values
healthcheck-mp          Checks that all instances are responding to Microprofile
Healthcheck requests with an UP response

Command list-healthcheck-services executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "`set-healthcheck-configuration`"

List-Http-Listeners

Lists the existing network listeners.

Synopsis

```
asadmin [asadmin-options] list-http-listeners [--help]
[target]
```

Description

The `list-http-listeners` subcommand lists the existing network listeners.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: Restricts the listing to network listeners for a specified target. Valid values are as follows:

`server`;; Lists the netowork listeners for the default server instance. This is the default value. configuration-name;; Lists the network listeners for the specified configuration. cluster-name;; Lists the network listeners for all server instances in the specified cluster. instance-name;; Lists the network listeners for the specified server instance.

Examples

Example 1 Listing Network Listeners

The following command lists all the network listeners for the server instance:

```
asadmin> list-http-listeners
```

```
http-listener-1
http-listener-2
admin-listener
Command list-http-listeners executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-http-listener,
- delete-http-listener,
- restart-http-listeners

List-iiop-Listeners

Lists the existing IIOP listeners.

Synopsis

```
asadmin [asadmin-options] list-iiop-listeners [--help]
[target]
```

Description

The `list-iiop-listeners` subcommand lists the existing IIOP listeners.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: This operand specifies the target for which the IIOP listeners are to be listed. Valid values are:

`server`;; Lists the listeners in the default server instance `server` and is the default

value. configuration_name;; Lists the listeners in the specified configuration. cluster_name;; Lists the listeners in the specified cluster. instance_name;; Lists the listeners in a particular server instance.

Examples

Example 1 Using the list-iiop-listeners subcommand

The following command lists all the IIOP listeners for the server instance:

```
asadmin> list-iiop-listeners
orb-listener-1
SSL
SSL_MUTUALAUTH
sample_iiop_listener
Command list-iiop-listeners executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-iiop-listener,
- delete-iiop-listener

List-Instances

Lists Payara Server instances in a domain.

Synopsis

```
asadmin [asadmin-options] list-instances [--help]
[--timeoutmsec timeout]
[--long={false|true} | --nostatus={false|true}]
[--standaloneonly={false|true} | target]
```

Description

The `list-instances` subcommand lists Payara Server instances in a domain. The list can be filtered by cluster, instance, node, or configuration.

The subcommand displays every Payara Server instance in the specified target, regardless of how each instance was created. For example, this subcommand lists instances that were created by using the create-instance subcommand and by using the create-local-instance subcommand.

By default, the subcommand indicates whether each instance that is listed is running. Options of this subcommand control the information that is displayed for each instance.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--timeoutmsec`**: The time limit in milliseconds for determining the status of instances. The default is 2,000, which is equivalent to 2 seconds. **`--long`**: **`-l`**: Specifies whether detailed information is displayed for each instance that is listed. The `--long` option and `--nostatus` option are mutually exclusive. If both options are specified in the same command, an error occurs.
Valid values are as follows:
`true`;; The following details are displayed for each instance that is listed:
* The name of the instance * The name of the host where the instance's node resides * The HTTP port on which the instance listens for administration requests * The process identifier (PID) of the instance process or -1 if the instance is not running * The name of the cluster of which the instance is a member, if any * The state of the instance, which is `running` or `not running`
When an instance is listed, some configuration changes in the domain administration server (DAS) for the instance might not have been applied to the instance itself. In this situation, the commands that are required to apply the changes are listed adjacent to the state of the instance. The maximum number of commands that are listed for an instance is 10. `false`;; Only the name of the instance and an indication of whether the instance is running are displayed (default). The length of time that the instance has been running is not displayed. **`--nostatus`**: Specifies whether information about

whether instances are running is suppressed.

The `--long` option and `--nostatus` option are mutually exclusive. If both options are specified in the same command, an error occurs.

Valid values are as follows:

`true`;; Information about whether instances are running is suppressed. Only the name of each instance is displayed. `false`;; Information about whether instances are running is displayed (default). **`--standaloneonly`**: Specifies whether only standalone instances are listed.

The `--standaloneonly` option and the target operand and are mutually exclusive. If both the `--standaloneonly` option and the target operand are specified in the same command, an error occurs.

Valid values are as follows:

`true`;; Only standalone instances are listed. `false`;; All instances in the specified target are listed (default).

Operands

target: Filters the list of Payara Server instances by specifying the target for which instances are listed.

The target operand and the `--standaloneonly` option are mutually exclusive. If both the target operand and the `--standaloneonly` option are specified in the same command, an error occurs.

Valid values are as follows:

`domain`;; Lists all instances in the domain (default). cluster-name;; Lists the instances that are members of the specified cluster. instance-name;; Lists only the specified instance. node-name;; Lists the instances that reside on the specified node. configuration-name;; Lists all instances whose configuration is defined by the specified named configuration.

Examples

Example 1 Listing Basic Information About All Payara Server Instances in a Domain

This example lists the name and status of all Payara Server instances in the current domain.

```
asadmin> list-instances
pmd-i-sj02 running
yml-i-sj02 running
pmd-i-sj01 running
yml-i-sj01 running
pmdsa1 not running

Command list-instances executed successfully.
```

Example 2 Listing Detailed Information About All Payara Server Instances in a Domain

This example lists detailed information about all Payara Server instances in the current domain.

```
asadmin> list-instances --long=true
NAME         HOST       PORT   PID    CLUSTER     STATE
pmd-i-sj01   sj01       24848  31310  pmdcluster   running
yml-i-sj01   sj01       24849  25355  ymlcluster   running
pmdsa1       localhost  24848  -1     ---          not running
pmd-i-sj02   sj02       24848  22498  pmdcluster   running
yml-i-sj02   sj02       24849  20476  ymlcluster   running
ymlsa1       localhost  24849  -1     ---          not running
Command list-instances executed successfully.
```

Example 3 Displaying the Status of an Instance

This example displays status of the instance `pmd-i-sj01`, which is running.

```
asadmin> list-instances pmd-i-sj01
pmd-i-sj01 running
Command list-instances executed successfully.
```

Example 4 Listing Only Standalone Instances in a Domain

This example lists only the standalone instances in the current domain.

```
asadmin> list-instances --standaloneonly=true
pmdsa1 not running
Command list-instances executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-instance,
- create-local-instance

List-Jacc-Providers

Enables administrators to list JACC providers defined for a domain.

Synopsis

```
asadmin [asadmin-options] list-jacc-providers [--help]
[target]
```

Description

The `list-jacc-providers` subcommand enables administrators to list the JACC providers defined for a domain. JACC providers are defined as `jacc-provider` elements in the `security-service` element in the domain's `domain.xml` file. JACC providers can be created using the Payara Server Admin Console or the `create-jacc-provider` subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: Specifies the target for which you want to list JACC providers. The following values are valid:
`server`;; Lists the JACC providers on the default server instance. This is the default value. configuration_name;; Lists the JACC providers in the specified configuration. cluster_name;; Lists the JACC providers on all server instances in the specified cluster. instance_name;; Lists the JACC providers on a specified server instance.

Examples

Example 1 Listing JACC providers

The following example shows how to list JACC providers for the default domain.

```
asadmin> list-jacc-providers
default
simple
testJACC
Command list-jacc-providers executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jacc-provider,
- delete-jacc-provider

List-Javamail-Resources

Lists the existing JavaMail session resources.

Synopsis

```
asadmin [asadmin-options] list-javamail-resources [--help] [target]
```

Description

The `list-javamail-resources` subcommand lists the existing JavaMail session resources.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: This operand specifies the target for which the JavaMail session resources are to be listed. Valid values are:
`server`;; Lists the resources for the default server instance. This is the default value.
`domain`;; Lists the resources for the domain. cluster_name;; Lists the resources for the specified cluster. instance_name;; Lists the resources for a particular server instance.

Examples

Example 1 Listing JavaMail Resources

This example lists the JavaMail session resources for the server instance.

```
asadmin> list-javamail-resources
mail/MyMailSession
Command list-javamail-resources executed successfuly.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-javamail-resource,
- delete-javamail-resource

List-Jdbc-Connection-Pools

Lists all JDBC connection pools.

Synopsis

```
asadmin [asadmin-options] list-jdbc-connection-pools [--help]
```

Description

The `list-jdbc-connection-pools` subcommand lists the current JDBC connection pools.

This subcommand is supported in the remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 Listing the JDBC Connection Pools

This example lists the existing JDBC connection pools.

```
asadmin> list-jdbc-connection-pools
sample_derby_pool
__TimerPool
Command list-jdbc-connection-pools executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jdbc-connection-pool,
- delete-jdbc-connection-pool

List-Jdbc-Resources

Lists all JDBC resources.

Synopsis

```
asadmin [asadmin-options] list-jdbc-resources [--help]
[target target]
```

Description

The `list-jdbc-resources` subcommand displays a list of the existing JDBC resources.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

`--target`: This operand specifies which JDBC resources you can list. Usage of this operand is optional. Valid values are:
`server`;; Lists the JDBC resources in the current server and is the default. `domain`;; Lists the JDBC resources in the current domain. cluster_name;; Lists the JDBC resources in a cluster. instance_name;; Lists the JDBC resources for a particular instance.

Examples

Example 1 Listing the JDBC Resources

This example lists the current JDBC resources.

```
asadmin> list-jdbc-resources
jdbc/DerbyPool
Command list-jdbc-resources executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jdbc-resource,
- delete-jdbc-resource

List-Jms-Hosts

Lists the existing JMS hosts.

Synopsis

```
asadmin [asadmin-options] list-jms-hosts [--help]
```

```
[--target target]
```

Description

The `list-jms-hosts` subcommand lists the existing Java Message Service (JMS) hosts for the JMS service.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Restricts the listing to JMS hosts for a specified target. Valid values are as follows:
`server`;; Lists the JMS hosts for the default server instance. This is the default value. configuration-name;; Lists the JMS hosts for the specified configuration. cluster-name;; Lists the JMS hosts for all server instances in the specified cluster. instance-name;; Lists the JMS hosts for the specified server instance.

Examples

Example 1 Listing all JMS hosts

The following subcommand lists the JMS hosts for the JMS service.

```
asadmin> list-jms-hosts server-config
default_JMS_host
MyNewHost
Command list-jms-hosts executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jms-host,

- delete-jms-host,
- jms-ping

List-Jms-Resources

Lists the JMS resources.

Synopsis

```
asadmin [asadmin-options] list-jms-resources [--help]
[--restype type]
[target]
```

Description

The `list-jms-resources` subcommand lists the existing Java Message Service (JMS) resources (destination and connection factory resources).

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--restype`**: The JMS resource type can be `jakarta.jms.Topic`, `jakarta.jms.Queue`, `jakarta.jms.ConnectionFactory`, `jakarta.jms.TopicConnectionFactory`, or `jakarta.jms.QueueConnectionFactory`.

Operands

target: Restricts the listing to resources for a specified target. Valid values are as follows:
`server`;; Lists the resources for the default server instance. This is the default value.
`domain`;; Lists the resources for the domain. cluster-name;; Lists the resources for every server instance in the specified cluster. instance-name;; Lists the resources for the specified server instance.

Examples

Example 1 Listing all JMS resources

The following subcommand lists all JMS resources.

```
asadmin> list-jms-resources
jms/Queue
jms/ConnectionFactory
jms/DurableConnectionFactory
jms/Topic
Command list-jms-resources executed successfully.
```

Example 2 Listing JMS resources of a specified type

The following subcommand lists all `jakarta.jms.ConnectionFactory` resources.

```
asadmin> list-jms-resources --restype jakarta.jms.ConnectionFactory
jms/ConnectionFactory
jms/DurableConnectionFactory
Command list-jms-resources executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jms-resource,
- delete-jms-resource

List-Jmsdest

Lists the existing JMS physical destinations.

Synopsis

```
asadmin [asadmin-options] list-jmsdest [--help]
[--desttype type]
[target]
```

Description

The `list-jmsdest` subcommand lists the Java Message Service (JMS) physical

destinations.

This subcommand is supported in remote mode only. Remote `asadmin` subcommands require a running domain administration server (DAS).

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--desttype`**: The type of JMS destination to be listed. Valid values are `topic` and `queue`.

Operands

target: Restricts the listing to physical destinations for a specified target. Valid values are as follows:
`server`;; Lists the physical destinations for the default server instance. This is the default value. configuration-name;; Lists the physical destinations in the specified configuration. cluster-name;; Lists the physical destinations for every server instance in the specified cluster. instance-name;; Lists the physical destinations for the specified server instance.

Examples

Example 1 Listing all physical destinations

The following subcommand lists all the physical destinations.

```
asadmin> list-jmsdest
PhysicalQueue
PhysicalTopic
Command list-jmsdest executed successfully.
```

Example 2 Listing all physical destinations of a specified type

The following subcommand lists all physical topics.

```
asadmin> list-jmsdest --desttype topic
PhysicalTopic
Command list-jmsdest executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jmsdest,
- delete-jmsdest,
- flush-jmsdest

List-Jndi-Entries

Browses and queries the JNDI tree.

Synopsis

```
asadmin [asadmin-options] list-jndi-entries [--help]
[--context context_name]
[target]
```

Description

Use this subcommand to browse and query the JNDI tree.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--context`**: The name of the JNDI context or subcontext. If context is not specified, all entries in the naming service are returned. If context (such as `ejb`) is specified, all those entries are returned.

Operands

target: This operand specifies the JNDI tree to browse.
Possible values are as follows:
`server`;; Browses the JNDI tree for the default Payara Server instance (default). The default instance is the domain administration server (DAS). `domain`;; Browses the JNDI

tree for the current domain. cluster-name;; Browses the JNDI tree for the specified cluster. instance-name;; Browses the JNDI tree for the specified Payara Server instance.

Examples

Example 1 Browsing the JNDI Tree

This example browses the JNDI tree for the default Payara Server instance.

```
asadmin> list-jndi-entries
java:global: com.sun.enterprise.naming.impl.TransientContext
jdbc: com.sun.enterprise.naming.impl.TransientContext
ejb: com.sun.enterprise.naming.impl.TransientContext
com.sun.enterprise.container.common.spi.util.InjectionManager:
com.sun.enterprise.container.common.impl.util.InjectionManagerImpl

Command list-jndi-entries executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jndi-resource,
- delete-jndi-resource,
- list-jndi-resources

List-Jndi-Resources

Lists all existing JNDI resources.

Synopsis

```
list-jndi-resources [--help] [target]
```

Description

The `list-jndi-resources` subcommand identifies all existing JNDI resources.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand.

Operands

target: The target for which the JNDI resources are to be listed. Valid values are as follows:

`server`;; Lists the JNDI resources on the default server instance. This is the default value. configuration-name;; Lists the JNDI resources for the specified configuration. cluster-name;; Lists the JNDI resources on all server instances in the specified cluster. instance-name;; Lists the JNDI resources on a specified server instance.

Examples

Example 1 Listing JNDI Resources

This example lists the JNDI resources on the default server instance.

```
asadmin> list-jndi-resources
jndi_resource1
jndi_resource2
jndi_resource3
Command list-jndi-resources executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jndi-resource,
- delete-jndi-resource

List-Jobs

Lists information about subcommands that were started using `asadmin --detach` or that contain progress information.

Synopsis

```
asadmin [asadmin-options] list-jobs [--help]
[job_id]
```

Description

The `list-jobs` subcommand lists information about subcommands that were started using the `asadmin` utility option `--detach` or that contain progress information. The `--detach` option detaches long-running subcommands and executes them in the background in detach mode.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

job_id: The ID of the job for which you want to list information.

Examples

Example 1 Checking Job Status

This example provides information about subcommands that were started using `asadmin --detach` or that contain progress information.

```
asadmin> list-jobs
JOB ID     COMMAND           STATE       EXIT CODE TIME OF COMPLETION
1          create-cluster    COMPLETED   SUCCESS   2013-02-15 16:16:16 PST
2          deploy            COMPLETED   FAILURE   2013-02-15 18:26:30 PST
Command list-jobs executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- attach,
- configure-managed-jobs

List-Jvm-Options

Lists options for the Java application launcher.

Synopsis

```
asadmin [asadmin-options] list-jvm-options [--help]
[--target target]
```

Description

The `list-jvm-options` subcommand displays a list of command-line options that are passed to the Java application launcher when Payara Server is started.

The options are managed by using the JVM Options page of the Administration Console or by using the `create-jvm-options` and `delete-jvm-options` subcommands.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which you are listing Java application launcher options.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Examples

Example 1 Listing the Java Application Launcher Options

This example lists the options that are used by the Java application launcher.

```
asadmin> list-jvm-options
-Djava.security.auth.login.config=${com.sun.aas.instanceRoot}/config/login.conf
-XX: LogVMOutput
-XX: UnlockDiagnosticVMOptions
-Dcom.sun.enterprise.config.config_environment_factory_class=
com.sun.enterprise.config.serverbeans.AppserverConfigEnvironmentFactory
-Djavax.net.ssl.keyStore=${com.sun.aas.instanceRoot}/config/keystore.jks
-XX:NewRatio=2
-DANTLR_USE_DIRECT_CLASS_LOADING=true
-Djava.security.policy=${com.sun.aas.instanceRoot}/config/server.policy
-Djdbc.drivers=org.apache.derby.jdbc.ClientDriver
-Djavax.net.ssl.trustStore=${com.sun.aas.instanceRoot}/config/cacerts.jks
-client
-Djava.ext.dirs=${com.sun.aas.javaRoot}/lib/ext${path.separator}${
com.sun.aas.javaRoot}/jre/lib/ext${path.separator}${com.sun.aas.instanceRoot}
/lib/ext${path.separator}${com.sun.aas.derbyRoot}/lib
-Xmx512m
-XX:MaxPermSize=192m
-XX:LogFile=${com.sun.aas.instanceRoot}/logs/jvm.log
Command list-jvm-options executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-jvm-options,
- delete-jvm-options

For more information about the Java application launcher, see the reference page for the operating system that you are using:

- Oracle Solaris and Linux: java - the Java application launcher (http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/java.html)
- Windows: java - the Java application launcher (http://docs.oracle.com/javase/6/docs/technotes/tools/windows/java.html)

List-Keystore-Entries

Displays entries in the specified domain keystore

Synopsis

```
asadmin list-keystore-entries [--help]
[--verbose={false|true}]
[--domainName=name]
[--domainDir=directory]
[--node=nodeName]
[--nodeDir=directory]
[--listener=listenername]
[--target=instanceName]
[alias]
```

Description

This command displays either all or a specific store entry from the target instance or listener's key store.

Options

`--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are listing its keystore entries. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. `--verbose`: Whether to print the full entry details. Defaults to `false` `--listener`: The name of the HTTP or IIOP listener to add the certificate to. `--domainname`: The name of the domain where the target instance exists.

+ Defaults to `domain1` or the existing domain if only one exists. `--domainDir`: The path to the directory containing the target domain.

+ Defaults to `as-install/glassfish/domains` `--node`: The name of the node where the target instance exists.

+ Defaults to `localhost-$domainname` `--nodeDir`: The path to the directory containing the target node.

+ Defaults to `as-install/glassfish/nodes`

Operands

alias: The alias name of the entry to list. If not provided then all entries in the keystore are listed.

Examples

Example 1 List all entries in the `production` domain's keystore

```
asadmin list-keystore-entries --domainName production
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- "SSL Certificate Management",
- "`generate-csr`",
- "`renew-self-signed-certificates`",
- "`add-to-keystore`",
- "`add-to-truststore`"

List-Libraries

Lists library JAR files on Payara Server.

Synopsis

```
asadmin [asadmin-options] list-libraries [--help]
[--type={common|ext|app}]
```

Description

The `list-libraries` subcommand lists library archive files on Payara Server.

The `--type` option specifies the library type and the Payara Server directory for which libraries are to be listed.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--type`**: Specifies the library type and the Payara Server directory for

which libraries are listed. Valid values are as follows:
`common`;; Lists the library files for the Common class loader directory, `domain-dir/lib`. This is the default. `ext`;; Lists the library files for the Java optional package directory, `domain-dir/lib/ext`. `app`;; Lists the library files for the application-specific class loader directory, `domain-dir/lib/applibs`. `war`;; Lists the library files for the war-packaged class loader directory, `domain-dir/lib/warlibs`
For more information about these directories, see classloading in the Payara Server Application section.

Examples

Example 1 Listing Libraries

This example lists the libraries in the application-specific class loader directory on the default server instance.

```
asadmin> list-libraries --type app
mylib.jar
xlib.jar
ylib.jar
zlib.jar
Command list-libraries executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- add-library,
- remove-library,
- classloading in the Payara Server Application Development section

List-Log-Attributes

Lists all logging attributes defined for a specified target in a domain.

Synopsis

```
asadmin [asadmin-options] list-log-attributes [--help]
[target]
```

Description

The `list-log-attributes` subcommand lists all logging attributes currently defined for the specified Payara Server domain or target within a domain. The values listed correspond to the values in the `logging.properties` file for the domain.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand.

Operands

target: Valid values are:

* `server` - The default server instance. This is the default value. * configuration_name - The name of a specific configuration. * cluster_name - The name of a target cluster. * instance_name - The name of a target server instance.

Examples

Example 1 Listing the Logger Attributes for a Domain

This example lists all loggers attributes for the default domain.

```
asadmin> list-log-attributes
com.sun.enterprise.server.logging.GFFileHandler.alarms <false>
com.sun.enterprise.server.logging.GFFileHandler.file \
<${com.sun.aas.instanceRoot}/logs/server.log>
com.sun.enterprise.server.logging.GFFileHandler.flushFrequency ¬
com.sun.enterprise.server.logging.GFFileHandler.formatter \
<com.sun.enterprise.server.logging.UniformLogFormatter>
com.sun.enterprise.server.logging.GFFileHandler.logtoConsole <false>
com.sun.enterprise.server.logging.GFFileHandler.maxHistoryFiles ¬
com.sun.enterprise.server.logging.GFFileHandler.retainErrorsStasticsForHours ¬
com.sun.enterprise.server.logging.GFFileHandler.rotationLimitInBytes
com.sun.enterprise.server.logging.GFFileHandler.rotationTimelimitInMinutes ¬
com.sun.enterprise.server.logging.SyslogHandler.useSystemLogging <false>
handlers <java.util.logging.ConsoleHandler>
java.util.logging.ConsoleHandler.formatter \
```

```
<com.sun.enterprise.server.logging.UniformLogFormatter>
java.util.logging.FileHandler.count ¬
java.util.logging.FileHandler.formatter <java.util.logging.XMLFormatter>
java.util.logging.FileHandler.limit
java.util.logging.FileHandler.pattern <%h/java%u.log>
log4j.logger.org.hibernate.validator.util.Version <warn>

Command list-log-attributes executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- collect-log-files,
- list-log-levels,
- rotate-log,
- set-log-attributes,
- set-log-levels,
- "administering-the-logging-service" in Payara Server General Administration section

List-Log-Levels

Lists the loggers and their log levels.

Synopsis

```
asadmin [asadmin-options] list-log-levels [--help]
[--target target]
```

Description

The `list-log-levels` subcommand lists the current Payara Server loggers and their log levels. This subcommand reports on all the loggers that are listed in the `logging.properties` file. In some cases, loggers that have not been created by the respective containers will appear in the list.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: The server, cluster or server instance for which log levels will be listed.

Operands

target: Valid values are:

* server_name - Default target is `server`. If no target is specified then log levels are listed for the server. * cluster_name - The name of a target cluster. * instance_name - The name of a target server instance.

Examples

Example 1 Listing the Log Levels

This example lists the existing loggers and indicates how their log levels are set.

```
asadmin> list-log-levels
java.util.logging.ConsoleHandler        <FINEST>
jakarta.enterprise.resource.corba <INFO>
jakarta.enterprise.resource.javamail      <INFO>
jakarta.enterprise.resource.jdo   <INFO>
jakarta.enterprise.resource.jms   <INFO>
jakarta.enterprise.resource.jta   <INFO>
jakarta.enterprise.resource.resourceadapter       <INFO>
jakarta.enterprise.resource.sqltrace      <FINE>
jakarta.enterprise.resource.webcontainer.jsf.application  <INFO>
jakarta.enterprise.resource.webcontainer.jsf.config       <INFO>
jakarta.enterprise.resource.webcontainer.jsf.context      <INFO>
jakarta.enterprise.resource.webcontainer.jsf.facelets     <INFO>
jakarta.enterprise.resource.webcontainer.jsf.lifecycle    <INFO>
jakarta.enterprise.resource.webcontainer.jsf.managedbean  <INFO>
jakarta.enterprise.resource.webcontainer.jsf.renderkit    <INFO>
jakarta.enterprise.resource.webcontainer.jsf.resource     <INFO>
jakarta.enterprise.resource.webcontainer.jsf.taglib       <INFO>
jakarta.enterprise.resource.webcontainer.jsf.timing       <INFO>
jakarta.enterprise.system.container.ejb   <INFO>
jakarta.enterprise.system.container.ejb.mdb       <INFO>
jakarta.enterprise.system.container.web   <INFO>
jakarta.enterprise.system.core.classloading       <INFO>
jakarta.enterprise.system.core.config     <INFO>
jakarta.enterprise.system.core    <INFO>
jakarta.enterprise.system.core.naming     <INFO>
jakarta.enterprise.system.core.security   <INFO>
jakarta.enterprise.system.core.selfmanagement     <INFO>
jakarta.enterprise.system.core.transaction        <INFO>
jakarta.enterprise.system <INFO>
jakarta.enterprise.system.tools.admin     <INFO>
```

```
jakarta.enterprise.system.tools.backup    <INFO>
jakarta.enterprise.system.tools.deployment        <INFO>
jakarta.enterprise.system.util    <INFO>
jakarta.enterprise.system.webservices.registry    <INFO>
jakarta.enterprise.system.webservices.rpc <INFO>
jakarta.enterprise.system.webservices.saaj        <INFO>
javax   <INFO>
javax.org.glassfish.persistence <INFO>
org.apache.catalina     <INFO>
org.apache.coyote       <INFO>
org.apache.jasper       <INFO>
org.glassfish.admingui  <INFO>
org.jvnet.hk2.osgiadapter       <INFO>
Command list-log-levels executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- collect-log-files,
- list-log-attributes,
- rotate-log,
- set-log-attributes,
- set-log-levels
- "administering-the-logging-service" in Payara Server General Administration section

List-Loggers

Lists existing loggers.

Synopsis

```
asadmin [asadmin-options] list-loggers [--help]
```

Description

The `list-loggers` subcommand lists the existing Payara Server loggers. Internal loggers are not listed. The `list-loggers` subcommand lists the logger name, subsystem, and description.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 Listing the Loggers

This example lists the logger name, subsystem, and description for each logger. Some lines of output are omitted from this example for readability.

```
asadmin> list-loggers
Logger Name                              Subsystem         Logger Description
...
jakarta.enterprise.monitoring              Monitoring        Monitoring Logger
jakarta.enterprise.system.core.ee          AS-CORE           Java EE Core Kernel
jakarta.enterprise.system.jmx              JMX               JMX System Logger
jakarta.enterprise.system.tools.admin      ADMIN             Administration Services
...
Command list-loggers executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-log-levels,
- set-log-file-format

List-Managed-Executor-Services

Lists managed executor service resources.

Synopsis

```
asadmin [asadmin-options] list-managed-executor-services [--help]
[target]
```

Description

The `list-managed-executor-services` subcommand lists managed executor service resources.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand.

Operands

target: Specifies the target for which managed executor service resources are to be listed. Valid targets are:

+ `server`;; Lists the resources on the default server instance. This is the default value. `domain`;; Lists the resources for the domain. cluster-name;; Lists the resources on all server instances in the specified cluster. instance-name;; Lists the resources on a specified server instance.

Examples

Example 1 Listing Managed Executor Service Resources

This example lists managed executor service resources on the default server instance.

```
asadmin> list-managed-executor-services
concurrent/__defaultManagedExecutorService
concurrent/Executor1
concurrent/Executor2
Command list-managed-executor-services executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,

- create-managed-executor-service,
- delete-managed-executor-service

List-Managed-Scheduled-Executor-Services

Lists managed scheduled executor service resources.

Synopsis

```
asadmin [asadmin-options] list-managed-scheduled-executor-services [--help]
[target]
```

Description

The `list-managed-scheduled-executor-services` subcommand lists managed scheduled executor service resources.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: Specifies the target for which managed scheduled executor service resources are to be listed. Valid targets are:

+ `server`;; Lists the resources on the default server instance. This is the default value. `domain`;; Lists the resources for the domain. cluster-name;; Lists the resources on all server instances in the specified cluster. instance-name;; Lists the resources on a specified server instance.

Examples

Example 1 Listing Managed Scheduled Executor Service Resources

This example lists managed scheduled executor service resources on the default server instance.

```
asadmin> list-managed-scheduled-executor-services
concurrent/__defaultManagedExecutorService
concurrent/ScheduledExecutor1
concurrent/ScheduledExecutor2
Command list-managed-scheduled-executor-services executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-managed-scheduled-executor-service,
- delete-managed-scheduled-executor-service

List-Managed-Thread-Factories

Lists managed thread factory resources.

Synopsis

```
asadmin [asadmin-options] list-managed-thread-factories [--help]
[target]
```

Description

The `list-managed-thread-factories` subcommand lists managed thread factory resources.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: Specifies the target for which managed thread factory resources are to be listed. Valid targets are:

+ `server`;; Lists the resources on the default server instance. This is the default value. `domain`;; Lists the resources for the domain. cluster-name;; Lists the resources on all server instances in the specified cluster. instance-name;; Lists the resources on a specified server instance.

Examples

Example 1 Listing Managed Thread Factory Resources

This example lists managed thread factory resources on the default server instance.

```
asadmin> list-managed-thread-factories
concurrent/__defaultManagedThreadFactory
concurrent/Factory1
concurrent/Factory2
Command list-managed-thread-factories executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-managed-thread-factory,
- delete-managed-thread-factory

List-Message-Security-Providers

Lists all security message providers for the given message layer.

Synopsis

```
asadmin [asadmin-options] list-message-security-providers [--help]
--layer message_layer
[target]
```

Description

The `list-message-security-providers` subcommand enables administrators to list all security message providers (`provider-config` sub-elements) for the given message layer (`message-security-config` element of `domain.xml`).

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--layer`**: The message-layer for which the provider has to be listed. The default value is `HttpServlet`.

Operands

target: Restricts the listing to message security providers for a specific target. Valid values include:
`server`;; Lists providers for the default server instance `server` and is the default value. `domain`;; Lists providers for the domain. cluster;; Lists providers for the server instances in the cluster. instance;; Lists providers for a particular server instance.

Examples

Example 1 Listing message security providers

The following example shows how to list message security providers for a message layer.

```
asadmin> list-message-security-providers
--layer SOAP
XWS_ClientProvider
ClientProvider
XWS_ServerProvider
ServerProvider
Command list-message-security-providers executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-message-security-provider,
- delete-message-security-provider

List-Modules

Lists Payara Server modules.

Synopsis

```
asadmin [asadmin-options] list-modules [--help]
```

Description

The `list-modules` subcommand displays a list of modules that are accessible to the Payara Server module subsystem. The version of each module is shown.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 Listing Payara Server Modules

This example provides a partial listing of modules that are accessible to the Payara Server module subsystem

```
asadmin> list-modules
List Of Modules
Module Status Report Begins
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.naming [380]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.grizzly.nucleus-grizzly-all
[292]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.phonehome-
bootstrap [332]], State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.shell [401]], State = [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.osgi-resource-locator [1]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.connectors.internal-api
[51]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.asadmin-audit
[19]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.docker [78]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.security.ee [351]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-
```

```
modules.requesttracing-core [337]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.batch.glassfish-batch-
connector [109]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.security [353]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.deployment.dol [79]], State
= [READY]
OSGiModuleImpl:: Bundle = [org.objectweb.asm.tree.analysis [21]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.healthcheck-
core [132]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-metrics [276]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.jmx-monitoring [240]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.orb.enabler [299]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.core.kernel [251]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.healthcheck-checker [131]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.util [6]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.resources.connector [338]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.cli [4]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.payara-
executor-service [319]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.deployment.common [74]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.connectors.runtime [52]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.notification-jms-core [291]], State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.configadmin [394]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.transaction.internal-api
[362]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.concurrent.connector [45]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.internal-api [147]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jms.core [236]], State =
[READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.api [139]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.weld-integration [389]],
State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.gogo.shell [399]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.grizzly.glassfish-grizzly-
extra-all [116]], State = [READY]
OSGiModuleImpl:: Bundle = [org.objectweb.asm.commons [22]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.hk2.config-types [48]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.concurrent.impl [46]], State
= [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.config-api [47]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.notification-
cdi-eventbus-core [288]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-openapi [278]], State = [READY]
```

```
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.gf-admingui-
connector [96]], State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.eventadmin [395]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.hazelcast-
bootstrap [127]], State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.gogo.command [397]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.util [44]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-config-extensions [264]], State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.scr [400]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.gf-web-connector [105]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.glassfish-mbeanserver
[118]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.gf-weld-connector
[106]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.environment-warning [88]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.opentracing-
adapter [295]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.healthcheck-metrics [134]], State = [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.runlevel [144]], State = [READY]
OSGiModuleImpl:: Bundle =
[fish.payara.server.internal.resourcebase.resources.nucleus-resources [293]], State
= [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.utils [145]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.rest-service [344]],
State = [READY]
OSGiModuleImpl:: Bundle = [org.objectweb.asm.util [24]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-fault-tolerance [270]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.monitoring-core
[286]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.transaction.jts [250]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.load-balancer.gf-load-
balancer-connector [103]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.healthcheck-
cpool [133]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.amx-core [7]], State
= [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.core.glassfish [121]], State
= [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jdbc.config [207]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.security.services [352]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.microprofile-
config-service [265]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ejb.http-admin [83]], State
= [READY]
OSGiModuleImpl:: Bundle = [jakarta.inject.jakarta.inject-api [172]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ha.hazelcast-store [126]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.notification-
eventbus-core [290]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
```

```
modules.payara-micro-service [324]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.transaction.jta [249]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.notification-
core [289]], State = [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.locator [143]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.core.jakartaee-kernel
[195]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.rest-monitoring-service [343]], State = [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.class-model [31]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.container-common
[68]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ejb.gf-ejb-connector [99]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.core.logging [258]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.hk2.config [140]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.deployment.autodeploy [72]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.healthcheck-
stuck [136]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.glassfish-naming
[119]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-healthcheck [272]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.asadmin-
recorder [20]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.flashlight.framework [95]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.hazelcast-eclipselink-coordination [128]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.resources.javamail-connector
[198]], State = [READY]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.osgi-adapter [313]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.webservices.connector
[384]], State = [READY]
OSGiModuleImpl:: Bundle = [org.objectweb.asm [25]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.connectors.gf-connectors-
connector [98]], State = [READY]
OSGiModuleImpl:: Bundle = [org.objectweb.asm.tree [23]], State = [READY]
OSGiModuleImpl:: Bundle =
[fish.payara.server.internal.security.jacc.provider.inmemory [150]], State =
[READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jms.gf-jms-connector [100]],
State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jdbc.runtime [208]], State =
[READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.fileinstall [396]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.glassfish-api [107]],
State = [READY]
OSGiModuleImpl:: Bundle = [org.apache.felix.gogo.runtime [398]], State = [READY]
OSGiModuleImpl:: Bundle = [org.eclipse.angus.mail [10]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.gf-restadmin-connector
[104]], State = [READY]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.webservices.jsr109-impl
[247]], State = [READY]

OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.stats77 [359]], State
```

```
= [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.payara-jsr107 [321]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [com.sun.istack.commons-runtime [148]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.corba.glassfish-corba-orb [113]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [com.fasterxml.jackson.core.jackson-annotations [151]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.media.jersey-media-json-binding
[222]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-jwt-auth [274]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.mail-api [178]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.weld-integration-
fragment [388]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.containers.jersey-container-
grizzly2-http [215]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [ [175]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.jboss.weld.osgi-bundle [390]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.cluster.ssh [35]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.parsson.jakarta.json [176]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.corba.rmic [345]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [com.fasterxml.jackson.core.jackson-databind [153]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.persistence.core [303]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.cluster.common [34]], State
= [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.containers.glassfish.jersey-gf-ejb
[219]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-connector [267]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.hibernate.validator.cdi [137]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.microprofile.jwt [273]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.tyrus.spi [371]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.persistence.glassfish-
oracle-jdbc-driver-packages [120]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.external.management-api [259]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [com.fasterxml.jackson.core.jackson-core [152]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.persistence-api [180]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.tyrus.core [369]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.enterprise.concurrent-api [167]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.glue [377]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.opentelemetry-
repackaged [294]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.corba.glassfish-corba-omgapi [112]], State
= [RESOLVED]
OSGiModuleImpl:: Bundle = [jboss-classfilewriter [203]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.deployment.javaee-full
[76]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.media.jersey-media-json-jackson
[223]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.osgi.util.function [311]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.apache.felix.bundlerepository [301]], State =
```

```
[RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.mvc-api [179]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.ext.cdi.jersey-cdi1x [212]], State
= [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.extras.payara-micro-boot
[322]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.jms-api [174]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.pfl.pfl-tf [331]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.trilead-ssh2-
repackaged [363]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.media.jersey-media-jaxb [221]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.servlet.jsp-api [185]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.war-util [372]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.wasp.wasp [373]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.websocket-api [190]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.embed-api [376]], State
= [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.api.payara-api [317]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.exousia [90]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.monitoring-console.api [285]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.activation-api [159]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.krazo.jersey [253]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.resource-api [181]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.glassfish-ee-api
[114]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.parsson.media [316]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.scattered-archive-api
[346]], State = [RESOLVED]
OSGiModuleImpl:: Bundle =
[fish.payara.server.internal.security.jaspic.provider.framework [197]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.ejb-api [164]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.jakarta-ee9-shim
[158]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [com.hazelcast [130]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.grizzly.npn-osgi [123]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.microprofile.rest.client [281]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [com.fasterxml.jackson.dataformat.jackson-dataformat-yaml
[156]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.deployment.transformer.deployment-
transformer-api [77]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.core [141]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ejb.ejb-client [80]], State
= [RESOLVED]
OSGiModuleImpl:: Bundle = [org.osgi.util.promise [312]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.ha [379]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jakarta.enterprise.concurrent [168]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.microprofile.metrics [275]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.metro-xmlsec-
repackaged [261]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.jboss.logging.jboss-logging [204]], State =
[RESOLVED]
```

```
OSGiModuleImpl:: Bundle = [com.fasterxml.woodstox.woodstox-core [391]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.gmbal.gmbal [122]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [com.fasterxml.jackson.module.jackson-module-jakarta-
xmlbind-annotations [157]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.persistence.jpa.jpql [306]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.metro.webservices-extra-jdk-packages
[385]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.servlet-api [184]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.pfl.pfl-basic [328]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.media.jersey-media-multipart
[226]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.connectors.inbound-runtime
[50]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [yasson [393]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.cdi-api-fragment [28]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [angus-activation [9]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.flashlight.flashlight-extra-
jdk-packages [94]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [javax.enterprise.deploy-api [169]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [com.sun.xml.bind.jaxb-osgi [201]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.validation.jakarta.validation-api [189]], State
= [RESOLVED]
OSGiModuleImpl:: Bundle = [javassist [200]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.krazo.core [252]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.core.glassfish-javassist-
packages [117]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.annotation-framework
[11]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.expressly [91]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.metro.webservices-osgi [386]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-telemetry [283]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.core.jersey-client [213]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.osgi.service.component [310]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [com.nimbusds.nimbus-jose-jwt [287]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.security.ssl-impl [358]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [GlassFish-Application-Common-Module [15]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.xml.bind-api [193]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.security.enterprise-api [182]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ejb.ejb-container [81]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.media.jersey-media-json-processing
[224]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.jaxrs-client-tracing [202]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.core.jersey-common [214]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.jvnet.mimepull [284]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.ext.jersey-entity-filtering [218]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.ext.cdi.jersey-cdi1x-transaction
[211]], State = [RESOLVED]
```

```
OSGiModuleImpl:: Bundle = [org.glassfish.pfl.pfl-dynamic [329]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-modules.jsr107-
repackaged [246]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.persistence.asm [302]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.corba.glassfish-corba-internal-api [111]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.external.aopalliance-repackaged [14]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.security.jacc-api [162]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.rest-client [341]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.microprofile.health [271]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.ws.rs-api [192]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.microprofile.config [263]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.el-api [165]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.security.auth.message-api [161]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.containers.jersey-container-
servlet-core [216]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.enterprise.cdi-api [166]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.extras [142]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.yaml.snakeyaml [355]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.ext.jersey-bean-validation [209]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.media.jersey-media-sse [227]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ejb.internal-api [84]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.servlet.jsp.jstl-api [186]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.websocket-client-api [191]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.ext.cdi.jersey-cdi1x-servlet
[210]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [com.fasterxml.jackson.dataformat.jackson-dataformat-toml
[154]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.hibernate.validator [138]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.tyrus.client [364]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.microprofile.openapi [277]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.batch.payara-jbatch [320]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [reactive-streams [334]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.orb.connector [298]], State
= [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.jcip-annotations [205]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.ant [12]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.tyrus.server [370]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.interceptor-api [173]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jakarta.security.enterprise [183]], State
= [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.core.glassfish-extra-jre-
packages [115]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.opentracing-
repackaged [297]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.jmxremote_optional-
repackaged [241]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [stax2-api [360]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.microprofile.opentracing [279]], State =
```

```
[RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.core [375]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.jline [234]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.core.jersey-server [233]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.tyrus.container-servlet [368]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.tiger-types [361]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [com.fasterxml.classmate [32]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.eclipse.microprofile.fault.tolerance [269]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.cdi-auth-roles [29]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.security.websecurity [382]],
State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.inject.jersey-hk2 [220]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.xml.ws-api [194]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.cli [374]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.metro.webservices-api-osgi [383]], State =
[RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-common [262]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.transaction-api [188]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.libpam4j-repackaged
[256]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.deployment.javaee-core
[75]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.ext.microprofile.jersey-mp-rest-
client [228]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.batch-api [163]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.annotation-api [160]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.jakarta.faces [171]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [org.glassfish.ha.ha-api [124]], State = [RESOLVED]
OSGiModuleImpl:: Bundle = [jakarta.enterprise.lang-model [170]], State = [RESOLVED]

OSGiModuleImpl:: Bundle = [io.projectreactor.reactor-core [335]], State = [NEW]
OSGiModuleImpl:: Bundle = [security-connector-oauth2-client [348]], State = [NEW]
OSGiModuleImpl:: Bundle =
[fish.payara.server.internal.security.webservices.security [387]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.faces-compat [92]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-common-
full-plugin [54]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-reference-
manual-plugin [66]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.payara-micro-cdi [323]], State = [NEW]
OSGiModuleImpl:: Bundle = [security-connector-oidc-client [349]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.osgi-platforms.osgi-
container [315]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.rest-monitoring-
plugin [342]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-jms-plugin
[62]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.jstl-connector [248]],
State = [NEW]
```

```
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.j-interop-
repackaged [149]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.web.jakarta.servlet.jsp.jstl [187]], State
= [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.payara-console-
extras [318]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.resources.javamail-runtime
[199]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-common
[55]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-ejb-lite-
plugin [58]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.core [5]], State =
[NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.persistence.gf-jpa-connector
[102]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.security.realm-stores
[336]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.appclient.gf-client-module [97]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.appclient.server.appclient-
connector [16]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.eventbus-notifier-
console-plugin [89]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.cluster.admin [33]], State =
[NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.tyrus.container-glassfish-cdi [365]],
State = [NEW]
OSGiModuleImpl:: Bundle = [com.sun.pkg.client [333]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.jaspic-servlet-utils [196]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-jca-plugin
[60]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.backup [26]], State =
[NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.connectors.work-management
[392]], State = [NEW]
OSGiModuleImpl:: Bundle = [com.fasterxml.jackson.dataformat.jackson-dataformat-xml
[155]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admin.launcher [254]], State
= [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.persistence.entitybean-
container [87]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.tyrus.container-glassfish-ejb [366]],
State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.pfl.pfl-basic-tools [327]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.ext.jersey-mvc-jsp [230]], State =
[NEW]
OSGiModuleImpl:: Bundle = [org.apache.bcel [27]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.appclient.server.appclient-
server-core [17]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.payara-rest-endpoints [325]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ejb.opentracing [85]], State
= [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.load-balancer.admin [257]],
State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.external.schema2beans [347]], State =
[NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.external.dbschema [70]], State = [NEW]
```

```
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-jdbc-plugin
[61]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.dataprovider [69]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-rest-client [282]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.media.jersey-media-moxy [225]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.healthcheck-
service-console-plugin [135]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.soteria.spi.bean.decorator.weld [357]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.connectors.admin [49]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.simple-glassfish-api
[354]], State = [NEW]
OSGiModuleImpl:: Bundle = [net.minidev.json-smart [244]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.eclipse.persistence.jpa [305]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.ldapbp-repackaged
[255]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.eclipse.persistence.oracle [309]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jms.admin [235]], State =
[NEW]
OSGiModuleImpl:: Bundle = [fish.payara.transformer.payara [93]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-config [266]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-concurrent-
plugin [56]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.jersey-mvc-connector
[229]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.hazelcast-ejb-timer [129]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.pfl.pfl-tf-tools [330]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.jmx-monitoring-
plugin [239]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.hk2.hk2 [146]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.resources.runtime [339]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-payara-
branding-plugin [64]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.ext.jersey-proxy-client [232]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.microprofile-
console-plugin [268]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.deployment.admin [71]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.rest-client-ssl [340]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.eclipse.persistence.dbws [304]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.ext.jersey-mvc [231]], State =
[NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.sse [381]], State =
[NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jms.gf-jms-injection [101]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.osgi-platforms.osgi-cli-
remote [314]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.webservices.soap-tcp [356]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.deployment.deployment-client
```

```
[73]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.common.amx-javaee [8]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-plugin-
service [65]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.gui-plugin-common
[378]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.persistence.jpa-container
[242]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.microprofile-opentracing [280]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.batch.glassfish-batch-
commands [108]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jdbc.admin [206]], State =
[NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.tyrus.container-grizzly-client [367]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.packager.antlr-repackaged
[13]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.eclipse.persistence.moxy [308]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.webservices.metro-glue
[260]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-web-plugin
[67]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-cluster-
plugin [53]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.payara-appserver-
modules.opentracing-jaxws [296]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.corba.glassfish-corba-csiv2-idl [110]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.appclient.acc-config [2]], State =
[NEW]
OSGiModuleImpl:: Bundle = [net.minidev.accessors-smart [3]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-jts-plugin
[63]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.jms-notifier-
console-plugin [238]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.orb.iiop [300]], State =
[NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.persistence.common [326]],
State = [NEW]
OSGiModuleImpl:: Bundle = [org.glassfish.jersey.containers.jersey-container-servlet
[217]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.jms.handlers [237]], State =
[NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.jspcaching-connector
[245]], State = [NEW]
OSGiModuleImpl:: Bundle = [org.eclipse.persistence.jpa.modelgen.processor [307]],
State = [NEW]
OSGiModuleImpl:: Bundle = [jakarta.jws-api [177]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-corba-
plugin [57]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.security.ejb.security [86]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.security.connectors.security-connectors-api
[350]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.console-ejb-plugin
[59]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.web.jsf-connector [243]],
State = [NEW]
```

```
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ha.ha-file-store [125]],
State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.security.appclient.security
[18]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.ejb.ejb-full-container
[82]], State = [NEW]
OSGiModuleImpl:: Bundle = [fish.payara.server.internal.admingui.cdieventbus-
notifier-console-plugin [30]], State = [NEW]
Module Status Report Ends
Command list-modules executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-commands,
- list-containers

List-Network-Listeners

Lists the existing network listeners.

Synopsis

```
asadmin [asadmin-options] list-network-listeners [--help]
[target]
```

Description

The `list-network-listeners` subcommand lists the existing network listeners.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: Restricts the listing to network listeners for a specified target. Valid values are as

follows:

`server`;; Lists the network listeners for the default server instance. This is the default value. configuration-name;; Lists the network listeners for the specified configuration. cluster-name;; Lists the network listeners for all server instances in the specified cluster. instance-name;; Lists the network listeners for the specified server instance.

Examples

Example 1 Listing Network Listeners

The following command lists all the network listeners for the server instance:

```
asadmin> list-network-listeners
admin-listener
http-listener-1
https-listener-2
Command list-network-listeners executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-network-listener,
- set-network-listener-configuration,
- delete-network-listener

List-Nodes-Config

Lists all Payara Server nodes that do not support remote communication in a domain.

Synopsis

```
asadmin [asadmin-options] list-nodes-config [--help]
[--long={false|true}]
```

Description

The `list-nodes-config` subcommand lists all Payara Server nodes that do not

support remote communication in a domain.

NOTE	To list all nodes in a domain regardless of the type of the node, run the list-nodes subcommand.

By default, the subcommand displays the following information for each node that is listed:

- The name of the node
- The type of the node, which is `CONFIG`
- The name of the host that the node represents

The `--long` option of the subcommand specifies whether the nodes are listed in long format. In long format, the following additional information about each node is displayed:

- The path to the parent of the base installation directory of Payara Server on the host that the node represents
- A comma-separated list of the names of the Payara Server instances that reside on the node

If the `--terse` option of the asadmin-1m utility is `true` and the `--long` option of the subcommand is `false`, the subcommand lists only the name of each node.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--long`**: **`-l`**: Specifies whether the nodes are listed in long format. Possible values are as follows:
`true`;; The nodes are listed in long format. `false`;; The nodes are listed in short format (default).

Examples

Example 1 Listing Payara Server Nodes

This example displays the name, type, and host for all Payara Server nodes that do not support remote communication in the domain `domain1`.

```
asadmin> list-nodes-config
localhost-domain1  CONFIG  localhost
devnode  CONFIG  localhost
Command list-nodes-config executed successfully.
```

Example 2 Listing Only the Names of Payara Server Nodes

This example uses the `--terse` option of the `asadmin` utility to list only the names of the Payara Server nodes that do not support remote communication in the domain `domain1`.

```
asadmin> list-nodes-config --terse=true
localhost-domain1
devnode
```

Example 3 Listing Payara Server Nodes in Long Format

This example lists the Payara Server nodes that do not support remote communication in the domain `domain1`in long format.

```
asadmin> list-nodes-config --long=true
NODE NAME           TYPE     NODE HOST   INSTALL DIRECTORY    REFERENCED BY
localhost-domain1   CONFIG   localhost   /export/payara7
devnode             CONFIG   localhost   /export/payara7   pmdsa1
Command list-nodes-config executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-node-config,

- create-node-ssh,
- delete-node-config,
- delete-node-ssh,
- list-nodes,
- list-nodes-ssh

List-Nodes-Ssh

lists all Payara Server nodes that support communication over SSH in a domain.

Synopsis

```
asadmin [asadmin-options] list-nodes-ssh [--help]
[--long={false|true}]
```

Description

The `list-nodes-ssh` subcommand lists all Payara Server nodes that support communication over secure shell (SSH) in a domain.

NOTE	To list all nodes in a domain regardless of the type of the node, run the list-nodes subcommand.

By default, the subcommand displays the following information for each node that is listed:

- The name of the node
- The type of the node, which is `SSH`
- The name of the host that the node represents

The `--long` option of the subcommand specifies whether the nodes are listed in long format. In long format, the following additional information about each node is displayed:

- The path to the parent of the base installation directory of Payara Server on the host that the node represents

- A comma-separated list of the names of the Payara Server instances that reside on the node

If the `--terse` option of the asadmin-1m utility is `true` and the `--long` option of the subcommand is `false`, the subcommand lists only the name of each node.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--long`**: **`-l`**: Specifies whether the nodes are listed in long format. Possible values are as follows:
`true`;; The nodes are listed in long format. `false`;; The nodes are listed in short format (default).

Examples

Example 1 Listing Payara Server Nodes

This example displays the name, type, and host for all Payara Server nodes that support communication over SSH in a domain.

```
asadmin> list-nodes-ssh
sj02  SSH  sj02.example.com
sj01  SSH  sj01.example.com
Command list-nodes-ssh executed successfully.
```

Example 2 Listing Only the Names of Payara Server Nodes

This example uses the `--terse` option of the `asadmin` utility to list only the names of the Payara Server nodes that support communication over SSH in a domain.

```
asadmin> list-nodes-ssh --terse=true
sj02
sj01
```

Example 3 Listing Payara Server Nodes in Long Format

This example lists the Payara Server nodes that support communication over SSH in a domain in long format.

```
asadmin> list-nodes-ssh --long=true
NODE NAME    TYPE    NODE HOST           INSTALL DIRECTORY    REFERENCED BY
sj02         SSH     sj02.example.com    /export/payara7   pmd-i-sj02, yml-i-sj02
sj01         SSH     sj01.example.com    /export/payara7   pmd-i-sj01, yml-i-sj01
Command list-nodes-ssh executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-node-config,
- create-node-ssh,
- delete-node-config,
- delete-node-ssh,
- list-nodes,
- list-nodes-config

List-Nodes

Lists all Payara Server nodes in a domain.

Synopsis

```
asadmin [asadmin-options] list-nodes [--help]
[--long={false|true}]
```

Description

The `list-nodes` subcommand lists all Payara Server nodes in a domain.

By default, the subcommand displays the following information for each node that is listed:

- The name of the node
    - The type of the node, which is one of the following types*: : **`CONFIG`**: The node does not support remote communication. **`SSH`**: The node supports communication over secure shell (SSH).
- The name of the host that the node represents

The `--long` option of the subcommand specifies whether the nodes are listed in long format. In long format, the following additional information about each node is displayed:

- The path to the parent of the base installation directory of Payara Server on the host that the node represents
- A comma-separated list of the names of the Payara Server instances that reside on the node

If the `--terse` option of the asadmin-1m utility is `true` and the `--long` option of the subcommand is `false`, the subcommand lists only the name of each node.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--long`**: **`-l`**: Specifies whether the nodes are listed in long format. Possible values are as follows:
`true`;; The nodes are listed in long format. `false`;; The nodes are listed in short format (default).

Examples

Example 1 Listing Payara Server Nodes

This example displays the name, type, and host for all Payara Server nodes in the domain `domain1`.

```
asadmin> list-nodes
localhost-domain1  CONFIG  localhost
sj02  SSH  sj02.example.com
sj01  SSH  sj01.example.com
devnode  CONFIG  localhost
Command list-nodes executed successfully.
```

Example 2 Listing Only the Names of Payara Server Nodes

This example uses the `--terse` option of the `asadmin` utility to list only the names of the Payara Server nodes in the domain `domain1`.

```
asadmin> list-nodes --terse=true
localhost-domain1
sj02
sj01
devnode
```

Example 3 Listing Payara Server Nodes in Long Format

This example lists the Payara Server nodes in the domain `domain1` in long format.

```
asadmin> list-nodes --long=true
NODE NAME           TYPE     NODE HOST          INSTALL DIRECTORY     REFERENCED BY
localhost-domain1   CONFIG   localhost          /export/payara7
sj02                SSH      sj02.example.com   /export/payara7    pmd-i2, yml-i2
sj01                SSH      sj01.example.com   /export/payara7    pmd-i1, yml-i1
devnode             CONFIG   localhost          /export/payara7    pmdsa1
Command list-nodes executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-node-config,
- create-node-ssh,
- delete-node-ssh,
- list-nodes-config,

- list-nodes-ssh

List-Notifiers

List all available notifiers to the server.

Synopsis

```
asadmin [asadmin-options] list-notifiers
[--help]
[--target target]
```

Description

This command can be used to list all available notifiers.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 List current notifiers

```
asadmin>  list-notifiers
Available Notifier Services:
        cdieventbus-notifier
        eventbus-notifier
        log-notifier
        jms-notifier

Command list-notifiers executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- notification-configure

List-Password-Aliases

Lists all password aliases.

Synopsis

```
asadmin [asadmin-options] list-password-aliases [--help]
```

Description

This subcommand lists all of the password aliases.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 Listing all password aliases

```
asadmin> list-password-aliases
jmspassword-alias
Command list-password-aliases executed successfully
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-password-alias,
- delete-password-alias,
- update-password-alias

List-Persistence-Types

Lists registered persistence types for HTTP sessions and SFSB instances.

Synopsis

```
asadmin [asadmin-options] list-persistence-types [--help]
--type={web|ejb}
```

Description

The `list-persistence-types` subcommand lists registered persistence types for HTTP sessions and stateful session bean (SFSB) instances. The built-in persistence types are `memory`, `file`, and `hazelcast`. The`memory` type does not apply to SFSB instances.

Other persistence types can be added using the `StrategyBuilder` class.

To set the persistence type for HTTP sessions, use the `set` subcommand to set the `persistence-type` attribute. For example:

```
asadmin> set c1-config.availability-service.web-container-availability.persistence-
type=file
```

To set the persistence type for SFSB instances without availability enabled, use the `set` subcommand to set the `sfsb-persistence-type` attribute. For example:

```
asadmin> set c1-config.availability-service.ejb-container-availability.sfsb-
persistence-type=file
```

To set the persistence type for SFSB instances with availability enabled, use the `set` subcommand to set the `sfsb-ha-persistence-type` attribute. For example:

```
asadmin> set
c1-config.availability-service.ejb-container-availability.sfsb-ha-persistence-type
=hazelcast
```

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--type`**: Specifies the type of sessions for which persistence types are

listed. Allowed values are as follows:

* `web` — Lists persistence types for HTTP sessions. * `ejb` — Lists persistence types for SFSB instances.

Examples

Example 1 Listing Persistence Types for HTTP Sessions

This example lists persistence types for HTTP sessions.

```
asadmin> list-persistence-types --type=web
memory
file
hazelcast
Command list-persistence-types executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- get,
- list,
- set,

List-Phone-Home

List the current status of the Phone Home service.

Synopsis

```
asadmin> list-phone-home
```

Description

The phone home data is presently used to gather usage statistics about the corresponding distribution, such as how long users typically run instances, and what versions they use. This command shows whether the phone home service is enabled or disabled.

Examples

Example 1 List Phone Home Service

```
asadmin>  list-phone-home
Phone Home Service is enabled

Command list-phone-home executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- disable-phone-home,
- enable-phone-home

List-Protocol-Filters

Lists the existing protocol filters.

Synopsis

```
asadmin [asadmin-options] list-protocol-filters [--help]
[--target server]
protocol-name
```

Description

The `list-protocol-filters` subcommand lists the existing protocol filters. This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **target**: Restricts the listing to protocol filters for a specified target. Valid values are as follows:
`server`;; Lists the protocol filters for the default server instance. This is the default value. configuration-name;; Lists the protocol filters for the specified configuration.

cluster-name;; Lists the protocol filters for all server instances in the specified cluster.

instance-name;; Lists the protocol filters for the specified server instance.

Operands

protocol-name: The name of the protocol for which to list protocol filters.

Examples

Example 1 Listing Protocol Filters

The following command lists all the protocol filters for the server instance:

```
asadmin> list-protocol-filters http1
http1-filter
https1-filter
Command list-protocol-filters executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-protocol-filter,
- delete-protocol-filter

List-Protocol-Finders

Lists the existing protocol finders.

Synopsis

```
asadmin [asadmin-options] list-protocol-finders [--help]
[--target server]
protocol-name
```

Description

The `list-protocol-finders` subcommand lists the existing protocol finders.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **target**: Restricts the listing to protocol finders for a specified target. Valid values are as follows:
`server`;; Lists the protocol finders for the default server instance. This is the default value. configuration-name;; Lists the protocol finders for the specified configuration. cluster-name;; Lists the protocol finders for all server instances in the specified cluster. instance-name;; Lists the protocol finders for the specified server instance.

Operands

protocol-name: The name of the protocol for which to list protocol finders.

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-protocol-finder,
- delete-protocol-finder

List-Protocols

Lists the existing protocols.

Synopsis

```
asadmin [asadmin-options] list-protocols [--help]
[target]
```

Description

The `list-protocols` subcommand lists the existing protocols.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: Restricts the listing to protocols for a specified target. Valid values are as follows:

`server`;; Lists the protocols for the default server instance. This is the default value. configuration-name;; Lists the protocols for the specified configuration. cluster-name;; Lists the protocols for all server instances in the specified cluster. instance-name;; Lists the protocols for the specified server instance.

Examples

Example 1 Listing Protocols

The following command lists all the protocols for the server instance:

```
asadmin> list-protocols
admin-listener
http-1
http-listener-1
http-listener-2
Command list-protocols executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-protocol,
- delete-protocol

List-Requesttraces

Retrieves a list of request tracing service traces.

Synopsis

```
asadmin [asadmin-options] list-requesttraces [--help]
[--first] value
[--historicTraces={true|false}]
```

Description

This command can be used to list a number of previous request trace messages from either the current or historic trace stores of the Request Tracing Service.

The request tracing messages are in JSON format, so can be parsed by log parsing tools as if it were in the following format:

```
{
   "traceSpans":[
      {
         "operationName":"processContainerRequest",
         "spanContext":{
            "spanId":"61fa3c1c-9f62-43fe-8e55-faba2ce77f92",
            "traceId":"13bdc5c0-7827-4106-ad32-0242fb75a732"
         },
         "startTime":"2018-03-15T12:36:07.391Z[Europe/London]",
         "endTime":"2018-03-15T12:36:07.560Z[Europe/London]",
         "traceDuration":"169000000",
         "spanTags":[
            {
               "Server":"server"
            },
            {
               "Domain":"domain1"
            }
         ]
      },
      {
         "operationName":"authenticateJaspic",
         "spanContext":{
            "spanId":"fb929e4d-1477-4841-a088-0d07888608a6",
            "traceId":"13bdc5c0-7827-4106-ad32-0242fb75a732"
         },
         "startTime":"2018-03-15T12:36:07.391Z[Europe/London]",
         "endTime":"2018-03-15T12:36:07.392Z[Europe/London]",
         "traceDuration":"1000000",
         "spanTags":[
            {
               "Context":""
            },
            {
               "AppContext":"__asadmin "
            },
            {
               "AuthResult":"true"
            },
            {
               "Principal":"admin"
```

```
            }
         ]
      },
      {
         "operationName":"processServletRequest",
         "spanContext":{
            "spanId":"94cc4722-0990-4230-aa34-f0e0f629b1d8",
            "traceId":"13bdc5c0-7827-4106-ad32-0242fb75a732"
         },
         "startTime":"2018-03-15T12:36:07.392Z[Europe/London]",
         "endTime":"2018-03-15T12:36:07.559Z[Europe/London]",
         "traceDuration":"167000000",
         "spanTags":[
            {
               "referer":"[http://localhost:4848/common/index.jsf]"
            },
            {
               "accept-language":"[en-GB,en;q=0.8,de;q=0.5,zh-CN;q=0.3]"
            },
            {
               "cookie":"[JSESSIONID=99ce446fd68ec4adb3a3aa61b087; treeForm_tree-
hi=treeForm:tree:nodes:hazelcast]"
            },
            {
               "QueryString":"bare=true"
            },
            {
               "Method":"GET"
            },
            {
               "URL":"http://localhost:4848/cluster/cluster/clusters.jsf"
            },
            {
               "accept":"[*/*]"
            },
            {
               "host":"[localhost:4848]"
            },
            {
               "connection":"[keep-alive]"
            },
            {
               "Class":"jakarta.faces.webapp.FacesServlet"
            },
            {
               "accept-encoding":"[gzip, deflate]"
            },
            {
               "user-agent":"[Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0]"
            }
         ],
         "references":[
            {
               "spanContext":{
                  "spanId":"61fa3c1c-9f62-43fe-8e55-faba2ce77f92",
                  "traceId":"13bdc5c0-7827-4106-ad32-0242fb75a732"
               },
               "relationshipType":"ChildOf"
            }
```

```
          ]
       }
    ]
}
```

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--first`**: Defaults to `false`. A limit for the number of Request Tracing messages to return in the output. **`--historicTraces`**: Defaults to `true`. Whether to get the traces from the historic trace store or not.

Examples

Example 1 Retrieve Non-historic traces

```
asadmin> list-requesttraces --historictraces false
```

Will give an output similar to the following:

```
Occurring Time            Elapsed Time  Traced Message
2018-03-15T12:36:07.391Z  169           {"traceSpans":[
{"operationName":"processContainerRequest","spanContext":{"spanId":"61fa3c1c-9f62-
43fe-8e55-faba2ce77f92","traceId":"13bdc5c0-7827-4106-ad32-
0242fb75a732"},"startTime":"2018-03-
15T12:36:07.391Z[Europe/London]","endTime":"2018-03-
15T12:36:07.560Z[Europe/London]","traceDuration":"169000000","spanTags":[{"Server":
"server"},{"Domain": "domain1"}]},
{"operationName":"authenticateJaspic","spanContext":{"spanId":"fb929e4d-1477-4841-
a088-0d07888608a6","traceId":"13bdc5c0-7827-4106-ad32-
0242fb75a732"},"startTime":"2018-03-
15T12:36:07.391Z[Europe/London]","endTime":"2018-03-
15T12:36:07.392Z[Europe/London]","traceDuration":"1000000","spanTags":[{"Context":
""},{"AppContext": "__asadmin "},{"AuthResult": "true"},{"Principal": "admin"}]},
{"operationName":"processServletRequest","spanContext":{"spanId":"94cc4722-0990-
4230-aa34-f0e0f629b1d8","traceId":"13bdc5c0-7827-4106-ad32-
0242fb75a732"},"startTime":"2018-03-
15T12:36:07.392Z[Europe/London]","endTime":"2018-03-
15T12:36:07.559Z[Europe/London]","traceDuration":"167000000","spanTags":[{"referer"
: "[http://localhost:4848/common/index.jsf]"},{"accept-language": "[en-
GB,en;q=0.8,de;q=0.5,zh-CN;q=0.3]"},{"cookie":
"[JSESSIONID=99ce446fd68ec4adb3a3aa61b087; treeForm_tree-
hi=treeForm:tree:nodes:hazelcast]"},{"QueryString": "bare=true"},{"Method":
"GET"},{"URL": "http://localhost:4848/cluster/cluster/clusters.jsf"},{"accept":
"[*/*]"},{"host": "[localhost:4848]"},{"connection": "[keep-alive]"},{"Class":
"jakarta.faces.webapp.FacesServlet"},{"accept-encoding": "[gzip, deflate]"},{"user-
agent": "[Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
Firefox/52.0]"}],"references":[{"spanContext":{"spanId":"61fa3c1c-9f62-43fe-8e55-
faba2ce77f92","traceId":"13bdc5c0-7827-4106-ad32-
0242fb75a732"},"relationshipType":"ChildOf"}]}
```

```
]}
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- set-requesttracing-configuration
- get-requesttracing-configuration

List-Resource-Adapter-Configs

Lists the names of the current resource adapter configurations.

Synopsis

```
asadmin [asadmin-options] list-resource-adapter-configs [--help]
[--raname raname] [--verbose {false|true}]
```

Description

This command lists the configuration information in the `domain.xml` for the connector module. It lists an entry called `resource-adapter-config` in the `domain.xml` file. If the `--raname` option is specified, only the resource adapter configurations for the specified connector module are listed.

This command is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--raname`**: Specifies the connector module name. **`--verbose`**: Lists the properties that are configured. Default value is false.

Examples

Example 1 Listing the Resource Adapter Configurations

This example lists the current resource adapter configurations.

```
asadmin> list-resource-adapter-configs
ra1
ra2
Command list-resource-adapter-configs executed successfully
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-resource-adapter-config,
- delete-resource-adapter-config

List-Resource-Refs

Lists existing resource references.

Synopsis

```
asadmin [asadmin-options] list-resource-refs [--help]
[target]
```

Description

The `list-resource-refs` subcommand lists all resource references in a cluster or an unclustered server instance. This effectively lists all the resources (for example, JDBC resources) available in the JNDI tree of the specified target.

The target instance or instances in the cluster need not be running or available for this subcommand to succeed.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the

subcommand.

Operands

target: The target for which you are listing the resource references. Valid targets are as follows:

`server`;; Lists the resource references for the default server instance and is the default target. cluster_name;; Lists the resource references for every server instance in the cluster. instance_name;; Lists the resource references for the named unclustered server instance.

Examples

Example 1 Listing Resource References for a Cluster

This example lists resource references for the cluster `cluster1`.

```
asadmin> list-resource-refs cluster1
jms/Topic
Command list-resource-refs executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-resource-ref,
- delete-resource-ref

List-Rest-Endpoints

Lists all the rest endpoints of a deployed web application

Synopsis

```
asadmin [asadmin-options] list-rest-endpoints [--help]
[--componentname=componentname]
[appname]
```

Description

Lists all the rest endpoints of a deployed web application.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: `-?`: Displays the help text for the subcommand.

`--componentname`: Name of the REST endpoint component to list.

Operands

`appname`: Name of the deployed application to list all of their rest endpoints.

Examples

Example 1 Listing all the rest endpoints for a deployed application

This example lists all the rest endpoints for the application `sse-producer`.

```
asadmin list-rest-endpoints sse-producer
GET     /sse-producer/rest/application.wadl
GET     /sse-producer/rest/sse/register
Command list-rest-endpoints executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

List-Secure-Admin-Internal-Users

Lists the user names that the Payara Server DAS and instances use to authenticate with each other and to authorize admin operations.

Synopsis

```
asadmin [asadmin-options] list-secure-admin-internal-users [--help]
[--long={false|true}]
[--output output]
[--header={false|true}]
[name]
```

Description

The `list-secure-admin-internal-users` subcommand lists the user names that the Payara Server DAS and instances use to authenticate with each other and to authorize admin operations.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--long`**: **`-l`**: Displays detailed information about each internal user. The default value is `false.` **`--output`**: **`-o`**: Displays specific details about each internal user. Use a comma-separated list to specify the details you want to display and their order. The values are case-insensitive.
Possible values are as follows:
`username`;; Displays the user name for the internal user. `password-alias`;; Displays the password alias for the internal user. **`--header`**: **`-h`**: Specifies whether column headings are displayed when the `--long` option is used. The default value is `true`. To suppress the headings, set the `--header` option to `false`.

Operands

name: The user name for the internal user for which you want to display details.

Examples

Example 1 Listing the User Name for Secure Admin

This example lists the user names that the Payara Server DAS and instances use to authenticate with each other and to authorize admin operations.

```
asadmin> list-secure-admin-internal-users

Command list-secure-admin-internal-users executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- enable-secure-admin,
- enable-secure-admin-internal-user

List-Secure-Admin-Principals

Lists the certificates for which Payara Server accepts admin requests from clients.

Synopsis

```
asadmin [asadmin-options] list-secure-admin-principals [--help]
[--long={false|true}]
[--output output]
[--header={false|true}]
[name]
```

Description

The `list-secure-admin-principals` subcommand lists the certificates for which Payara Server accepts admin requests from clients.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--long`**: **`-l`**: Displays detailed information about the certificates for which Payara Server accepts admin requests from clients. The default value is `false`. **`--output`**: **`-o`**: Displays specific details about the certificates for which Payara Server accepts admin requests from clients. Use a comma-separated list to specify the details you want to display and their order. The values are case-insensitive.
Possible values are as follows:
`DN`;; Displays the distinguished name (DN) of each certificate. **`--header`**: **`-h`**: Specifies whether column headings are displayed when the `--long` option is used. The default value is `true`. To suppress the headings, set the `--header` option to `false`.

Operands

name: The distinguished name of the certificate, specified as a comma-separated list in quotes. For example: `"CN=system.amer.oracle.com,OU=GlassFish,O=Oracle Corporation,L=Santa Clara,ST=California,C=US"`.

Examples

Example 1 Listing the Certificates

This example lists the certificates for which Payara Server accepts admin requests from clients.

```
asadmin> list-secure-admin-principals
CN=localhost,OU=Payara,O==Payara Foundation,L=Great Malvern,ST=Worcestershire,C=UK
CN=localhost-instance,OU=Payara,O=Payara Foundation,L=Great Malvern,ST
=Worcestershire,C=UK
Command list-secure-admin-principals executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- enable-secure-admin,
- enable-secure-admin-principal

List-Sub-Components

Lists EJB or servlet components in a deployed module or module of a deployed application.

Synopsis

```
asadmin [asadmin-options] list-sub-components [--help]
[--type type]
[--appname appname] [--resources]
modulename
```

Description

The `list-sub-components` subcommand lists EJB or servlet components in a deployed module or in a module of a deployed application. If a module is not specified, all modules are listed. The `--appname` option functions only when the specified module is stand-alone. To display a specific module in an application, you must specify the module name with the `--appname` option.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--type`**: Specifies the type of component to be listed. The options are `ejbs` and `servlets`. If nothing is specified, then all of the components are listed. **`--appname`**: Identifies the name of the application. This option is required when the desired output is the subcomponents of an embedded module of a deployed application.
The name can include an optional version identifier, which follows the name and is separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. For more information about module and application versions, see module-and-application-versions in Payara Server Application Deployment section.
`--resources`: Lists the application-scoped resources for each subcomponent.

Operands

modulename: Specifies the name of the module containing the subcomponent.
The name can include an optional version identifier, which follows the name and is separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. For more information about module and application versions, see module-and-application-versions in Payara Server Application Deployment section.

Examples

Example 1 Listing Subcomponents

This example lists the subcomponents of the `MEjbApp` application within the `mejb.jar` module.

```
asadmin> list-sub-components --appname MEjbApp mejb.jar
MEJBBean <StatelessSessionBean>
Command list-sub-components executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- disable,
- enable,

List-Supported-Cipher-Suites

Enables administrators to list the cipher suites that are supported and available to a specified Payara Server target.

Synopsis

```
asadmin [asadmin-options] list-supported-cipher-suites [--help]
[--target target]
```

Description

The `list-supported-cipher-suites` subcommand enables administrators to list the cipher suites that are supported and available to a specified Payara Server target. The cipher suites that may be available in addition to the default SSL/TLS providers that are bundled with Payara Server packages will vary depending on the third-party provider.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which you want to list cipher suites. The following values are valid:
`server`;; Lists the cipher suites for the default server instance. This is the default value.
configuration_name;; Lists the cipher suites for the specified configuration.
cluster_name;; Lists the cipher suites for all server instances in the specified cluster.
instance_name;; Lists the cipher suites for a specified server instance.

Examples

Example 1 Listing cipher suites

The following example shows how to list cipher suites for the default domain.

```
asadmin> list-supported-cipher-suites
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_WITH_AES_128_CBC_SHA
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA

Command list-supported-cipher-suites executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m

List-System-Properties

Lists the system properties of the domain, configuration, cluster, or server instance.

Synopsis

```
asadmin [asadmin-options] list-system-properties [--help]
[target]
```

Description

Shared or clustered server instances will often need to override attributes defined in their referenced configuration. Any configuration attribute in a server instance can be overridden through a system property of the corresponding name. This list-system-properties subcommand lists the system properties of a domain, configuration, cluster, or server instance.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: This restricts the listing to system properties for a specific target. Valid values are:
domain;; Lists the system properties defined for the domain. configuration_name;; Lists the system properties for the named configuration as well as those the cluster inherits from the domain. cluster_name;; Lists the system properties defined for the named cluster as well as those the cluster. inherits from its configuration and the domain. instance_name;; Lists the system properties defined for the named server instance as well as those the server inherits from its cluster (if the instance is clustered), its configuration, and the domain.

Examples

Example 1 Listing System Properties

This example lists the system properties on `localhost`.

```
asadmin> list-system-properties
http-listener-port=1088
Command list-system-properties executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-system-properties,
- delete-system-property

List-Threadpools

Lists all the thread pools.

Synopsis

```
asadmin [asadmin-options] list-threadpools [--help]
target
```

Description

The `list-threadpools` subcommand lists the Payara Server thread pools.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: This operand specifies the target for which you are listing thread pools. This operand is required.
Valid values are as follows:
`server`;; Lists the thread pools for the default Payara Server instance `server`. configuration-name;; Lists the thread pools for the named configuration. cluster-name;; Lists the thread pools for every instance in the cluster. instance-name;; Lists the thread pools for a particular instance.

Examples

Example 1 Listing Thread Pools

This example lists the current thread pools for the default instance `server`.

```
asadmin> list-threadpools server
admin-thread-pool
http-thread-pool
thread-pool-1
Command list-threadpools executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-threadpool,
- delete-threadpool

List-Timers

Lists all persistent timers owned by server instance(s).

Synopsis

```
asadmin [asadmin-options] list-timers [--help]
[target]
```

Description

The `list-timers` subcommand lists the persistent timers owned by a specific server instance or a cluster of server instances.

This command is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: The target is either a standalone server instance or a cluster. If the target is the stand-alone instance, then the number of timers owned by the instance is listed. If the target is a cluster, then the number of timers owned by each instance in the cluster is listed. The default target is `server`, the default server instance.

Examples

Example 1 Listing Current Timers in a Server Instance

This example lists persistent timers in a particular standalone server instance. There is one currently active timer set.

```
asadmin> list-timers server
1

The list-timers command was executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- migrate-timers
- Using the Timer Service in The Java EE 8 Tutorial
- ejb-timer-service in the Payara Server Application Development section

List-Transports

Lists the existing transports.

Synopsis

```
asadmin [asadmin-options] list-transports [--help]
[target]
```

Description

The `list-transports` subcommand lists the existing transports.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

target: Restricts the listing to transports for a specified target. Valid values are as follows:

`server`;; Lists the transports for the default server instance. This is the default value. configuration-name;; Lists the transports for the specified configuration. cluster-name;; Lists the transports for all server instances in the specified cluster. instance-name;; Lists the transports for the specified server instance.

Examples

Example 1 Listing Transports

The following command lists all the transports for the server instance:

```
asadmin> list-transports
http1-trans
tcp
Command list-transports executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-transport,
- delete-transport

List-Truststore-Entries

Displays entries in the specified domain trust store

Synopsis

```
asadmin list-truststore-entries [--help]
[--verbose={false|true}]
[--domainName=name]
[--domainDir=directory]
[--node=nodeName]
[--nodeDir=directory]
[--listener=listenername]
[--target=instanceName]
[alias]
```

Description

This command displays either all or a specific store entry from the target instance or listener's trust store.

Options

`--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are listing its truststore entries. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. `--verbose`: Whether to print the full entry details. Defaults to `false` `--listener`: The name of the HTTP or IIOP listener to add the certificate to. `--domainname`: The name of the domain where the target instance exists.

+ Defaults to `domain1` or the existing domain if only one exists. `--domainDir`: The path to the directory containing the target domain.

+ Defaults to `as-install/glassfish/domains` `--node`: The name of the node where the target instance exists.

+ Defaults to `localhost-$domainname` `--nodeDir`: The path to the directory containing the target node.

+ Defaults to `as-install/glassfish/nodes`

Operands

alias: The alias name of the entry to list. If not provided then all entries in the truststore are listed.

Examples

Example 1 List all entries in the `production` domain's truststore

```
asadmin list-keystore-entries --domainName production
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- "SSL Certificate Management",
- "`generate-csr`",
- "`renew-self-signed-certificates`",
- "`add-to-keystore`",
- "`add-to-truststore`"

List-Virtual-Servers

Lists the existing virtual servers.

Synopsis

```
asadmin [asadmin-options] list-virtual-servers [--help]
[target]
```

Description

The `list-virtual-servers` subcommand lists the existing virtual servers.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand.

Operands

target: Restricts the listing to virtual servers for a specified target. Valid values are as

follows:

`server`;; Lists the virtual servers for the default server instance. This is the default value. configuration-name;; Lists the virtual servers for the specified configuration. cluster-name;; Lists the virtual servers for all server instances in the specified cluster. instance-name;; Lists the virtual servers for the specified server instance.

Examples

Example 1 Listing Virtual Servers

The following command lists all the virtual servers for the server instance:

```
asadmin> list-virtual-servers
server
__asadmin
Command list-virtual-servers executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-virtual-server,
- delete-virtual-server

List-Web-Context-Param

Lists servlet context-initialization parameters of a deployed web application or module.

Synopsis

```
asadmin [asadmin-options] list-web-context-param [--help]
[--name=context-param-name] application-name[/module]
```

Description

The `list-web-context-param` subcommand lists the servlet context-initialization parameters of one of the following items:

- A deployed web application
- A web module in a deployed Jakarta EE application

The application must already be deployed. Otherwise, an error occurs.

The `list-web-context-param` subcommand lists only parameters that have previously been set by using the set-web-context-param subcommand. The subcommand does not list parameters that are set only in the application's deployment descriptor.

For each parameter, the following information is displayed:

- The name of the parameter
- The value to which the parameter is set
- The value of the `--ignoreDescriptorItem` option of the `set-web-context-param` subcommand that was specified when the parameter was set
- The description of the parameter or `null` if no description was specified when the parameter was set

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--name`**: The name of the servlet context-initialization parameter that is to be listed. If this option is omitted, all parameters of the application that have previously been set are listed.

Operands

application-name: The name of the application. This name can be obtained from the Administration Console or by using the list-applications subcommand.
The application must already be deployed. Otherwise, an error occurs. **module**: The relative path to the module within the application's enterprise archive (EAR) file. The path to the module is specified in the `module` element of the application's `application.xml` file.

module is required only if the servlet context-initialization parameter applies to a web module of a Jakarta EE application. If specified, module must follow application-name, separated by a slash (/).
For example, the `application.xml` file for the `myApp` application might specify the following web module:

+

```
<module>
    <web>
        <web-uri>myWebModule.war</web-uri>
    </web>
</module>
```

```
The module would be specified as the operand of this command as
`myApp/myWebModule.war`.
```

Examples

Example 1 Listing Servlet Context-Initialization Parameters for a Web Application

This example lists all servlet context-initialization parameters of the web application `basic-ezcomp` that have been set by using the `set-web-context-param` subcommand. Because no description was specified when the `javax.faces.PROJECT_STAGE` parameter was set, `null` is displayed instead of a description for this parameter.

```
asadmin> list-web-context-param basic-ezcomp
javax.faces.STATE_SAVING_METHOD = client ignoreDescriptorItem=false
//The location where the application's state is preserved
javax.faces.PROJECT_STAGE = null ignoreDescriptorItem=true //null
Command list-web-context-param executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,

- list-applications,
- set-web-context-param,
- unset-web-context-param

List-Web-Env-Entry

Lists environment entries for a deployed web application or module.

Synopsis

```
asadmin [asadmin-options] list-web-env-entry [--help]
[--name=env-entry-name] application-name[/module]
```

Description

The `list-web-env-entry` subcommand lists the environment entries for one of the following items:

- A deployed web application
- A web module in a deployed Jakarta EE application

The application must already be deployed. Otherwise, an error occurs.

The `list-web-env-entry` subcommand lists only entries that have previously been set by using the set-web-env-entry subcommand. The subcommand does not list environment entries that are set only in the application's deployment descriptor.

For each entry, the following information is displayed:

- The name of the entry
- The Java type of the entry
- The value to which the entry is set
- The value of the `--ignoreDescriptorItem` option of the `set-web-env-entry` subcommand that was specified when the entry was set
- The description of the entry or `null` if no description was specified when the entry was set

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--name`**: The name of the environment entry that is to be listed. The name is a JNDI name relative to the `java:comp/env` context. The name must be unique within a deployment component. If this option is omitted, all environment entries that have previously been set for the application are listed.

Operands

application-name: The name of the application. This name can be obtained from the Administration Console or by using the list-applications subcommand.
The application must already be deployed. Otherwise, an error occurs. **module**: The relative path to the module within the application's enterprise archive (EAR) file. The path to the module is specified in the `module` element of the application's `application.xml` file.
module is required only if the environment entry applies to a web module of a Jakarta EE application. If specified, module must follow application-name, separated by a slash (/).
For example, the `application.xml` file for the `myApp` application might specify the following web module:

```
<module>
    <web>
        <web-uri>myWebModule.war</web-uri>
    </web>
</module>
```

```
The module would be specified as the operand of this command as
`myApp/myWebModule.war`.
```

Examples

Example 1 Listing Environment Entries for a Web Application

This example lists all environment entries that have been set for the web application `hello` by using the `set-web-env-entry` subcommand. Because no description was

specified when the `Hello Port` environment entry was set, `null` is displayed instead of a description for this entry.

```
asadmin> list-web-env-entry hello
Hello User (java.lang.String) = techscribe ignoreDescriptorItem=false
 //User authentication for Hello appplication
Hello Port (java.lang.Integer) = null ignoreDescriptorItem=true //null

Command list-web-env-entry executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- list-applications,
- set-web-env-entry,
- unset-web-env-entry

List

Lists configurable or monitorable elements.

Synopsis

```
asadmin [asadmin-options] list [--help]
[--monitor={false|true}]
[dotted-parent-attribute-name]
```

Description

The `list` subcommand lists configurable and monitorable attributes of Payara Server.

The output of the `list` subcommand is a list of the dotted names that represent individual server components and subsystems. For example, `server.applications.web-module`. After you know the particular component or subsystem, you can then use the `get` subcommand to access any attributes, and the `set` subcommand to modify configurable attributes.

The following rules apply to dotted names in a `list` subcommand:

NOTE	Characters that have special meaning to the shell or command interpreter, such as * (asterisk), should be quoted or escaped as appropriate to the shell, for example, by enclosing the argument in quotes. In multimode, quotes are needed only for arguments that include spaces, quotes, or backslash.

- Any `list` subcommand that has a dotted name that is not followed by a wildcard (*) lists the current node's immediate children. For example, the following command lists all immediate children belonging to the server node:

```
asadmin> list server
```

- Any `list` subcommand that has a dotted name followed by a wildcard(*) lists a hierarchical tree of child nodes from the current node. For example, the following command lists all child nodes of applications and their subsequent child nodes, and so on:

```
asadmin> list server.applications.*
```

- Any `list` subcommand that has a dotted name preceded or followed by a wildcard () **of the form *dotted name or dottedname** lists all nodes and their child nodes that match the regular expression created by the provided matching pattern.

For detailed information about dotted names, see the dotted-names help page.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--monitor`**: **`-m`**: Defaults to false. If set to false, the configurable attribute values are returned. If set to true, the monitorable attribute values are returned.

Operands

dotted-parent-element-name: Configurable or monitorable element name

Examples

Example 1 Listing Dotted Names of Configurable Elements

This example lists the elements that can be configured.

```
asadmin> list *
applications
configs
configs.config.server-config
configs.config.server-config.admin-service
configs.config.server-config.admin-service.das-config
configs.config.server-config.admin-service.jmx-connector.system
configs.config.server-config.admin-service.property.adminConsoleContextRoot
configs.config.server-config.admin-service.property.adminConsoleDownloadLocation
configs.config.server-config.admin-service.property.ipsRoot
configs.config.server-config.ejb-container
configs.config.server-config.ejb-container.ejb-timer-service
configs.config.server-config.http-service
configs.config.server-config.http-service.access-log
configs.config.server-config.http-service.virtual-server.__asadmin
configs.config.server-config.http-service.virtual-server.server
configs.config.server-config.iiop-service
configs.config.server-config.iiop-service.iiop-listener.SSL
configs.config.server-config.iiop-service.iiop-listener.SSL.ssl
configs.config.server-config.iiop-service.iiop-listener.SSL_MUTUALAUTH
configs.config.server-config.iiop-service.iiop-listener.SSL_MUTUALAUTH.ssl
configs.config.server-config.iiop-service.iiop-listener.orb-listener-1
configs.config.server-config.iiop-service.orb
configs.config.server-config.java-config
configs.config.server-config.jms-service
configs.config.server-config.jms-service.jms-host.default_JMS_host
...
property.administrative.domain.name
resources
resources.jdbc-connection-pool.DerbyPool
resources.jdbc-connection-pool.DerbyPool.property.DatabaseName
resources.jdbc-connection-pool.DerbyPool.property.Password
resources.jdbc-connection-pool.DerbyPool.property.PortNumber
resources.jdbc-connection-pool.DerbyPool.property.User
resources.jdbc-connection-pool.DerbyPool.property.connectionAttributes
resources.jdbc-connection-pool.DerbyPool.property.serverName
resources.jdbc-connection-pool.__TimerPool
resources.jdbc-connection-pool.__TimerPool.property.connectionAttributes
resources.jdbc-connection-pool.__TimerPool.property.databaseName
resources.jdbc-resource.jdbc/__TimerPool
resources.jdbc-resource.jdbc/__default
servers
servers.server.server
servers.server.server.resource-ref.jdbc/__TimerPool
servers.server.server.resource-ref.jdbc/__default
system-applications
```

```
Command list executed successfully.
```

Example 2 Listing Attributes of a Configurable Element

This example lists the attributes of the web container.

```
asadmin> list configs.config.server-config.web-container
configs.config.server-config.web-container
configs.config.server-config.web-container.session-config
Command list executed successfully.
```

Example 3 Listing Dotted Names of Monitorable Objects

This example lists the names of the monitorable objects that are enabled for monitoring.

```
asadmin> list --monitor *
server.jvm
server.jvm.class-loading-system
server.jvm.compilation-system
server.jvm.garbage-collectors
server.jvm.garbage-collectors.Copy
server.jvm.garbage-collectors.MarkSweepCompact
server.jvm.memory
server.jvm.operating-system
server.jvm.runtime
server.network
server.network.admin-listener
server.network.admin-listener.connections
server.network.admin-listener.file-cache
server.network.admin-listener.keep-alive
server.network.admin-listener.thread-pool
server.network.http-listener-1
server.network.http-listener-1.connections
server.network.http-listener-1.file-cache
server.network.http-listener-1.keep-alive
server.network.http-listener-1.thread-pool
server.transaction-service
Command list executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,

- get,
- set,
- dotted-names

Login

Logs you into a domain.

Synopsis

```
asadmin [asadmin-options] login [--help]
```

Description

The purpose of the `login` subcommand is to ease domain administration by letting you log into a particular domain. If Payara Server domains are created on various machines (locally), you can run the `asadmin` utility from any of these machines and manage domains located elsewhere (remotely). This is especially useful when a particular machine is chosen as an administration client that manages multiple domains and servers.

The `login` subcommand prompts you for the administrator user name and password. After successful login, the `.asadminpass` file is created in your home directory. (This is the same file that is modified when you run the `create-domain` subcommand with the `--savelogin` option.) The literal host name is stored, and no resolution with the DNS is attempted. If a domain is being administered from other machines, it is sufficient to run the `login` subcommand once. You do not need to specify the `asadmin` utility options `--user` and `--passwordfile` when you run additional remote subcommands on that domain. After you have logged into a domain, you still need to provide the host and port for any subsequent remote subcommands unless you chose the default values for `--host` (localhost) and `--port` (4848) options.

Subsequent use of same subcommand with the same parameters will result in overwriting the contents of the `.asadminpass` file for the given administration host and port. You can decide to overwrite the file or to reject such a login.

Login information is saved permanently and can be used across multiple domain restarts.

There is no `logout` subcommand. If you want to log in to another domain, run the `login` subcommand and specify new values for the `asadmin` utility options `--host` and `--port`.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 Logging Into a Domain on a Remote Machine

This example logs into a domain located on another machine. Options are specified before the `login` subcommand.

```
asadmin --host foo --port 8282 login
Please enter the admin user name>admin
Please enter the admin password>

Trying to authenticate for administration of server at host [foo]
and port [8282] ...
Login information relevant to admin user name [admin] for host [foo]
and admin port [8282] stored at [/.asadminpass] successfully.
Make sure that this file remains protected. Information stored in this
file will be used by asadmin commands to manage associated domain.
```

Example 2 Logging Into a Domain on the Default Port of Localhost

This example logs into a domain on `mylhost` on the default port. Options are specified before the `login` subcommand.

```
asadmin --host myhost login
Please enter the admin user name>admin
Please enter the admin password>
Trying to authenticate for administration of server
at host [myhost] and port [4848] ...
An entry for login exists for host [myhost] and port [4848], probably
from an earlier login operation.
Do you want to overwrite this entry (y/n)?y
Login information relevant to admin user name [admin] for host [myhost]
```

```
and admin port [4848] stored at [/home/joe/.asadminpass] successfully.
Make sure that this file remains protected. Information stored in this
file will be used by asadmin commands to manage associated domain.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-domain,
- delete-domain

Migrate-Timers

Moves EJB timers when a clustered instance was stopped or has crashed.

Synopsis

```
asadmin [asadmin-options] migrate-timers [--help]
[--target target_server_name]
server_name
```

Description

The `migrate-timers` subcommand moves EJB timers to a specified server when a server instance stops or crashes, if automatic timer migration is not enabled in the cluster configuration.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This is the target server instance. If this option is not specified, then DAS will find a server instance or multiple server instances. A migration notification will be sent to the selected server instances.

Operands

server_name: This is the server instance on which the timers are currently located. This server instance should not be running during the migration process.

Examples

Example 1 Migrating Timers

This example shows how to migrate timers from the server named `instance1` to a server named `instance2`.

```
asadmin> migrate-timers --target instance2 instance1
This command was successfully executed.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- list-timers

Monitor

Displays monitoring data for commonly used components and services.

Synopsis

```
asadmin [asadmin-options] monitor [--help]
--type type
[--interval interval]
[--filter filter]
instance-name
```

Description

The `monitor` subcommand displays statistics for commonly monitored Payara Server components and services. The `--type` option must be used to specify the object for which statistics are to be displayed.Data is displayed continuously in a tabular form, or the data can be displayed at a particular time interval by using the `--interval` option.

Before a given component or service can be monitored, monitoring must be enabled (set to HIGH or LOW) for the component or service by using the Administration Console, the enable-monitoring subcommand, or the set subcommand.

The monitor subcommand has options for filtering the results and capturing the output in a Comma Separated Values (CSV) file. The output appears in a table format. To view the legend of the table header, type `h`.

NOTE	This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--type`**: The component or service to monitor. This option is required. No default value is defined.
`httplistener`;; For this type, the attribute `server.monitoring-service.module-monitoring-levels.http-service` must be set to LOW or HIGH.
Displays the following statistics for the HTTP listener service: : **`ec`**: The total number errors in the processing of HTTP requests. **`mt`**: The longest response time (in milliseconds) for the processing of a single HTTP request. **`pt`**: The total amount of time (in milliseconds) that the HTTP listener service has spent in processing HTTP requests. **`rc`**: The total number of requests that the HTTP listener service has processed. `jvm`;; For this type, the attribute `server.server-config.monitoring-service.module-monitoring-levels.jvm` must be set to LOW or HIGH.
Displays the following statistics for the Virtual Machine for the Java platform (Java Virtual Machine or JVM machine): **`UpTime`**: The number of milliseconds that the JVM machine has been running since it was last started. **`min`**: The initial amount of memory (in bytes) that the JVM machine requests from the operating system for memory management during startup. **`max`**: The maximum amount of memory that can be used for memory management. **`low`**: Retained for compatibility with other releases. **`high`**: Retained for compatibility with other releases. **`count`**: The amount of memory (in bytes)

that is guaranteed to be available for use by the JVM machine. `webmodule`;; For this type, the attribute `server.server-config.monitoring-service.module-monitoring-levels.web-container` must be set to LOW or HIGH.
Displays the following statistics for all deployed web modules: : **`asc`**: The number of currently active sessions. **`ast`**: The total number of sessions that are currently active or have been active previously. **`rst`**: The total number of rejected sessions. **`st`**: The total number of sessions that have been created. **`ajlc`**: The number of currently active JavaServer Pages (JSP) technology pages that are loaded. **`mjlc`**: The maximum number of JSP technology pages that were active at any time simultaneously. **`tjlc`**: Total number of JSP technology pages that have been loaded. **`aslc`**: The number of currently active Java servlets that are loaded. **`mslc`**: The maximum number of Java servlets that were active at any time simultaneously. **`tslc`**: The total number of Java servlets that have been loaded. **`--interval`**: The interval in seconds before capturing monitoring attributes. The interval must be greater than 0. The monitoring attributes are displayed on `stdout` until you type Control-C or `q`. The default value is 30. **`--filter`**: Do not specify this option. This option is retained for compatibility with earlier releases. If you specify this option, a syntax error does not occur. Instead, the subcommand runs successfully and displays a warning message that the option is ignored.

Operands

instance-name: The server instance for which to view monitoring data. The default value is `server`.

Examples

Example 1 Displaying Monitoring Statistics by Interval

This example displays monitoring data for the JVM machine every 2000 seconds.

```
asadmin> monitor --type=jvm --interval 2000 server

JVM Monitoring
UpTime(ms)                Heap and NonHeap Memory(bytes)
current                   min         max         low         high        count
957843                    29523968    188284928   0           0           60370944

Command monitor executed successfully.
```

Example 2 Filtering the Monitoring Data

This example uses the filter option to show `http-listener-1` statistics.

```
asadmin> monitor --type httplistener --filter http-listener-1 server

HTTP Listener Monitoring: http-listener-1
br   bs   c200 c2xx c302 c304 c3xx c400 c401 c403 c404 c4xx c503 c5xx coc  co
ctc  ctb  ec   moc  mst  mt   mtm  mst  pt   rc
0    0    0    0    0    3    3    0    0    0    0    0    0    0    0    0
2    0    0    1    20   20   2    2    6    3
```

To see the legend for the table headings, type `h`.

```
*******************************************************************************
*******
* br   = Cumulative value of the Bytes received by each of the Request Processors
*
* bs   = Cumulative value of the Bytes sent by each of the Request Processors
*
* c200 = Number of responses with a status code equal to 200
*
* c2xx = Number of responses with a status code in the 2xx range
*
* c302 = Number of responses with a status code equal to 302
*
* c304 = Number of responses with a status code equal to 304
*
* c3xx = Number of responses with a status code in the 3xx range
*
* c400 = Number of responses with a status code equal to 400
*
* c401 = Number of responses with a status code equal to 401
*
* c403 = Number of responses with a status code equal to 403
*
* c404 = Number of responses with a status code equal to 404
*
* c4xx = Number of responses with a status code equal to 4xx
*
* c504 = Number of responses with a status code equal to 504
*
* c5xx = Number of responses with a status code equal to 5xx
*
* coc  = Number of open connections
*
* co   = Number of responses with a status code outside the 2xx, 3xx, 4xx, and 5xx
range *
* ctc  = Number of request processing threads currently in the listener thread pool
*
* ctb  = Number of request processing threads currently in use in the listener
thread    *
*        pool serving requests
*
* ec   = Number of responses with a status code equal to 400
```

```
*
* moc  = Maximum number of open connections
*
* mst  = Minimum number of request processing threads that will be created at
listener   *
*        startup time and maintained as spare threads above the current thread
count     *
* mt   = Maximum number of request processing threads that are created by the
listener   *
* mtm  = Provides the longest response time for a request - not a cumulative value,
but  *
*        the largest response time from among the response times
*
* pt   = Cumulative value of the times taken to process each request. The
processing     *
*        time is the average of request processing times over the request count
*
* rc   = Cumulative number of the requests processed so far
*
*****************************************************************************
*******
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- enable-monitoring,
- disable-monitoring,
- set
- monitoring
- administering-the-monitoring-service in the Payara Server General Administration section

Monitoring

Reviewing the runtime state of components and services deployed in Payara Server.

Description

Monitoring is the process of reviewing the statistics of a system to improve performance or solve problems. By monitoring the state of various components and services deployed in Payara Server, performance bottlenecks can be identified, failures

can be anticipated, and runtime standards can be established and observed. Data gathered by monitoring can also be useful in performance tuning and capacity planning.

The Payara Server monitoring service is enabled by default, that is, the `monitoring-enabled` attribute of the `monitoring-service` element is set to true. Once the monitoring service is enabled, a deployed module can then be enabled for monitoring by setting its monitoring level to `HIGH` or `LOW` (default is `OFF`).

Monitoring can be configured dynamically by using the Administration Console or the `enable-monitoring`/`disable-monitoring` and `set-monitoring-service-configuration` subcommands. The `set` subcommand can also be used with dotted names to enable or disable monitoring. However, a server restart is required for changes made by using the `set` subcommand to take effect.

Monitoring data can be viewed by using the Administration Console or by using the subcommands of the `asadmin` utility.

- The `monitor` subcommand displays monitoring data for a given type, similar to the UNIX `top` command. The data is presented at given intervals.
- The `list` and `get` subcommands display comprehensive data. Both use dotted names to specify monitorable objects.

Alternate tools for monitoring Payara Server components and services include JConsole and the REST interface.

The Monitoring Scripting Client or DTrace Monitoring can be used to start the available monitoring probes. Using these tools is helpful in identifying performance issues during runtime.

TIP	Monitoring Scripting Client or DTrace Monitoring are only usable if their modules are present.

See Also

- monitor,
- enable-monitoring,
- disable-monitoring,
- set-monitoring-service-configuration,
- list,
- get,
- set,
- dotted-names,
- asadmin-1m,
- administering-the-monitoring-service in the Payara Server General Administration section

Multimode

Allows multiple subcommands to be run while preserving environment settings and remaining in the asadmin utility.

Synopsis

```
asadmin [asadmin-options] multimode [--help]
[--file filename]
[--printprompt={true|false}] [--encoding encode]
```

Description

The `multimode` subcommand processes `asadmin` subcommands sequentially in a single session. The command-line interface prompts for a subcommand, runs that subcommand, displays the results of that subcommand, and then prompts for the next subcommand. All the `asadmin` options set in multimode apply to subsequent commands until the multimode session is exited. You exit `multimode` by typing `exit`, `quit`, or Ctrl-D.

You can use the `export` subcommand to set your environment, or use the`unset` subcommand to remove environment variables from the multimode environment.

You can also provide subcommands by passing a previously prepared list of subcommands from a file or standard input (pipe). When you use a file, you can include comment lines in the file by entering the hash symbol (`#`) as the first character of the line.

You can invoke `multimode` from within a multimode session. When you exit the second multimode environment, you return to your original multimode environment.

All the remote `asadmin` utility options can be supplied when invoking the `multimode` subcommand. The settings will apply as defaults for all subcommands that are run within the multimode session. For a list of the `asadmin` utility options, see the asadmin-1m help page.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--file`**: **`-f`**: Reads the subcommands in the specified file. **`--printprompt`**: Controls printing of the `asadmin` prompt. By default, this option is set to the same value as the `--interactive asadmin` utility option. Normally you will not need to specify this option. Default is true. **`--encoding`**: Specifies the character set for the file to be decoded. By default, the system character set is used.

Examples

Example 1 Starting a Multimode Session

This example starts a multimode session where: `%` is the system prompt.

```
% asadmin multimode
asadmin>
```

You can also start a multimode session by typing `asadmin` without options or subcommands at the system prompt.

Example 2 Running Multiple Commands From a File

This example runs a sequence of subcommands from the `commands_file.txt` file.

```
% asadmin multimode --file commands_file.txt
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- export,
- unset,

Notification-Configure

Configures the notification service.

Synopsis

```
asadmin [asadmin-options] notification-configure
[--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
```

Description

This command can be used to configure the behaviour of the log notifier in Payara Server.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the notification service. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--enabled`**: Defaults to `false`. Enables or disables the notification service **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without

a server restart.

Examples

Example 1 Enables the notification service

```
asadmin> notification-configure --enabled=true --dynamic=true
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-log-notifier-configuration,
- get-jms-notifier-configuration,
- get-eventbus-notifier-configuration,
- get-cdieventbus-notifier-configuration

Osgi-Shell

Provides interactive access to the Apache Felix Gogo remote shell for the execution of OSGi shell commands.

Synopsis

```
asadmin [asadmin-options] osgi-shell [--help]
[--file file]
[--printprompt={false|true}]
[--encoding encoding]
```

Description

The `osgi-shell` subcommand provides interactive access to the Apache Felix Gogo remote shell for the execution of OSGi shell commands. The remote shell is provided with Payara Server and used to administer and inspect the OSGi runtime.

OSGi shell commands are executed on the server and results are printed on the client. Because the shell is interactive, no operands are accepted. Scripting is supported, which

means that multiple commands can be executed in sequence from a text file.

A related subcommand is the `osgi` subcommand, which passes a single command to the remote shell for execution. Results are returned by the `asadmin` utility. For more information about the `osgi` subcommand, see the osgi help page.

This subcommand is supported in local mode only. Unlike other local subcommands, however, the domain administration server (DAS) and the server instance whose shell is being accessed must be running.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--file`**: **`-f`**: Reads the commands in the specified file. **`--printprompt`**: Controls printing of the shell prompt. The default value is `true`, which means the shell prompt is displayed. **`--encoding`**: Specifies the character set for the file to be decoded. By default, the system character set is used.

Examples

Example 1 Listing Apache Felix Gogo Remote Shell Commands

This example lists Apache Felix Gogo remote shell commands.

Some lines of output are omitted from this example for readability.

```
asadmin> osgi-shell
Use "exit" to exit and "help" for online help.
gogo$ help
felix:bundlelevel
felix:cd
felix:frameworklevel
felix:headers
felix:help
felix:inspect
felix:install
felix:lb
felix:log
felix:ls
...
gogo:cat
gogo:each
gogo:echo
```

```
gogo:format
gogo:getopt
gogo:gosh
gogo:grep
...
gogo$
```

Example 2 Running a Remote Shell Command

This example runs the Felix Remote Shell Command `lb` without any arguments to list all installed OSGi bundles.

Some lines of output are omitted from this example for readability.

```
asadmin> osgi-shell
Use "exit" to exit and "help" for online help.
gogo$ lb
START LEVEL 2
ID|State      |Level|Name
 0|Active     |    0|System Bundle
 1|Active     |    1|Metro Web Services API OSGi Bundle
 2|Active     |    1|jakarta.annotation API
 3|Active     |    1|jaxb-api
...
gogo$
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- osgi

Osgi

Delegates the command line to the Apache Felix Gogo remote shell for the execution of OSGi shell commands.

Synopsis

```
asadmin [asadmin-options] osgi [--help]
[--session session]
[--session-id session-id]
[--instance instance]
```

```
[command_line]
```

Description

The `osgi` subcommand delegates the command line to the Apache Felix Gogo remote shell for the execution of OSGi shell commands. Commands are executed by the remote shell and results are returned by the `asadmin` utility. The remote shell is provided with Payara Server and used to administer and inspect the OSGi runtime.

Multiple command-line sessions can be created. Use the `--session` and `--session -id` options to run commands in a specific command-line session. If no session is specified, a new session is created to run commands and closed when execution completes.

A related subcommand is the `osgi-shell` subcommand, which enables you to run multiple commands from a file or run commands interactively. For more information about the `osgi-shell` subcommand, see the osgi-shell help page.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the `osgi` subcommand. **`--session`**: Performs command-line session operations. Valid values are:
`new`;; Creates a new session and returns a session ID. `stop`;; Stops the session with the specified session ID. `list`;; Lists all active sessions. `execute`;; Runs a command in the session with the specified session ID. **`--session-id`**: Specifies the session ID for command-line session operations. **`--instance`**: Specifies the server instance to which the command is being delegated.
The default is the domain administration server (DAS). The DAS must be running to run a command on another instance.

Operands

command_line: The complete command-line syntax as provided for commands in the

Apache Felix Gogo remote shell.

Examples

Example 1 Listing Apache Felix Gogo Remote Shell Commands

This example lists the Apache Felix Gogo remote shell commands that can be used with the `osgi` subcommand.

Some lines of output are omitted from this example for readability.

```
asadmin> osgi help
felix:bundlelevel
felix:cd
felix:frameworklevel
felix:headers
felix:help
felix:inspect
felix:install
felix:lb
felix:log
felix:ls
felix:refresh
felix:resolve
...
gogo:cat
gogo:each
gogo:echo
gogo:format
gogo:getopt
gogo:gosh
gogo:grep
...
Command osgi executed successfully.
```

Example 2 Running a Remote Shell Command

This example runs the Felix Remote Shell Command `lb` without any arguments to list all installed OSGi bundles.

Some lines of output are omitted from this example for readability.

```
asadmin> osgi lb
START LEVEL 2
ID|State      |Level|Name
 0|Active     |    0|System Bundle
 1|Active     |    1|Metro Web Services API OSGi Bundle
 2|Active     |    1|jakarta.annotation API
 3|Active     |    1|jaxb-api
```

```
...
Command osgi executed successfully.
```

Example 3 Running Commands That Create and Target a Specific Command-Line Session

This example runs commands that create and target a specific command-line session.

Some lines of output are omitted from this example for readability.

```
asadmin> osgi --session new
9537e570-0def-4f2e-9c19-bc8f51a8082f
...
asadmin> osgi --session list
9537e570-0def-4f2e-9c19-bc8f51a8082f
...
asadmin> osgi --session execute --session-id 9537e570-0def-4f2e-9c19-bc8f51a8082f
lb

START LEVEL 2
ID|State      |Level|Name
 0|Active     |    0|System Bundle
 1|Active     |    1|Metro Web Services API OSGi Bundle
 2|Active     |    1|jakarta.annotation API
 3|Active     |    1|jaxb-api
...
asadmin> osgi --session stop --session-id 9537e570-0def-4f2e-9c19-bc8f51a8082f

Command osgi executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- osgi-shell

Package-Appclient

Packs the application client container libraries and jar files.

Synopsis

```
package-appclient
```

Description

Use the `package-appclient` command to pack the application client container libraries and jar files into an `appclient.jar` file, which is created in the `as-install/lib` directory on the machine on which Payara Server is installed. The `appclient.jar` file provides an application client container package targeted at remote hosts that do not contain a server installation.

After copying the `appclient.jar` file to a remote location, unjar or unzip it to get a set of libraries and jar files in the `appclient` directory under the current directory.

After unjarring on the client machine, modify `appclient/glassfish/config/asenv.conf` (`asenv.bat` for Windows) as follows:

- set `AS_WEBSERVICES_LIB` to path-to-appclient`/appclient/lib`
- set `AS_IMQ_LIB` to path-to-appclient`/appclient/mq/lib`
- set `AS_INSTALL` to path-to-appclient`/appclient`
- set `AS_JAVA` to your JDK 7 or JRE 7 home directory
- set `AS_ACC_CONFIG` to path-to-appclient`/appclient/glassfish/domains/domain1/config/sun-acc.xml`

Modify `appclient/glassfish/domains/domain1/config/sun-acc.xml` as follows:

- Ensure the `DOCTYPE` file references path-to-appclient`/appclient/glassfish/lib/dtds`
- Ensure that `target-server` address attribute references the server machine.
- Ensure that `target-server` port attribute references the ORB port on the remote machine.
- Ensure that `log-service` references a log file; if the user wants to put log messages to a log file.

To use the newly installed application client container, you must do the following:

- Obtain the application client files for your target application, including the generated yourAppClient.jar file.
- Execute the `appclient` utility: `appclient -`client yourAppClient.jar

Attributes

See `attributes` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Unstable

See Also

- appclient

Ping-Connection-Pool

Tests if a connection pool is usable.

Synopsis

```
asadmin [asadmin-options] ping-connection-pool [--help]
pool_name
[--appname application [--modulename module]
```

Description

The `ping-connection-pool` subcommand tests if an existing JDBC or connector connection pool is usable . For example, if you create a new JDBC connection pool for an application that is expected to be deployed later, the JDBC pool is tested with this subcommand before deploying the application.

Before testing availability of a connection pool, you must create the connection pool with authentication and ensure that the server or database is started.

This subcommand is supported in remote mode only.

Application Scoped Resources

The `ping-connection-pool` subcommand can target resources that are scoped to a

specific application or module, as defined in the `glassfish-resources.xml` or `payara-resources.xml` for the Payara Server domain.

IMPORTANT	glassfish-resources.xml is deprecated and support will be removed in a future release. It is recommended to use payara-resources.xml instead.

- To reference the `jndi-name` for an application scoped resource, perform the lookup using the `java:app` prefix.
- To reference the `jndi-name` for a module scoped resource, perform the lookup using the `java:module` prefix.

The `jndi-name` for application-scoped-resources or module-scoped-resources are specified using the format `java:app/`jdbc/myDataSource` or `java:module/`jdbc/myModuleLevelDataSource`.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--appname`**: Name of the application in which the application scoped resource is defined. **`--modulename`**: Name of the module in which the module scoped resource is defined.

Operands

pool_name: Name of the connection pool to be reinitialized.

Examples

Example 1 Contacting a Connection Pool

This example tests to see if the connection pool named `DerbyPool` is usable.

```
asadmin> ping-connection-pool DerbyPool
Command ping-connection-pool executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-connector-connection-pool,
- create-jdbc-connection-pool,
- delete-connector-connection-pool,
- delete-jdbc-connection-pool,
- list-connector-connection-pools,
- list-jdbc-connection-pools

Ping-Node-Ssh

Tests if a node that is enabled for communication over SSH is usable.

Synopsis

```
asadmin [asadmin-options] ping-node-ssh [--help]
[--validate={false|true}] node-name
```

Description

The `ping-node-ssh` subcommand tests if a node that is enabled for communication over secure shell (SSH) is usable. This subcommand requires secure shell (SSH) to be configured on the machine where the domain administration server (DAS) is running and on the machine where the node resides. You may run this command from any machine that can contact the DAS.

If the node is usable, the subcommand displays a confirmation that the subcommand could connect to the node through SSH. This confirmation includes the name of the host that the node represents.

Optionally, the subcommand can also validate the node to determine if the asadmin-1m utility can run on the host that the node represents. To validate a node, the

subcommand runs the version subcommand. If the node is valid, the subcommand displays the version that the `version` subcommand returns.

The node that is specified as the operand of this subcommand must be enabled for communication over SSH. If the node is not enabled for communication over SSH, an error occurs. To determine whether a node is enabled for communication over SSH, use the list-nodes subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--validate`**: Specifies whether the subcommand validates the node. Possible values are as follows:
`true`;; The node is validated. `false`;; The node is not validated (default).

Operands

node-name: The name of the node to test. The node must be enabled for communication over SSH. Otherwise, an error occurs.

Examples

Example 1 Testing if an SSH-Enabled Node Is Usable

This example tests if the SSH-enabled node `sj03-node` is usable.

```
asadmin> ping-node-ssh sj03-node
Successfully made SSH connection to node sj03-node (sj03.example.com)
Command ping-node-ssh executed successfully.
```

Example 2 Validating an SSH-Enabled Node

This example validates the SSH-enabled node `adc-node`.

```
asadmin> ping-node-ssh --validate=true adc-node
Successfully made SSH connection to node adcnode (adc.example.com)
GlassFish version found at /export/payara:
Using locally retrieved version string from version class.
```

```
Version = Payara Server 7.2.0 #badassfish (build 00)
Command version executed successfully.
Command ping-node-ssh executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-node-ssh,
- delete-node-ssh,
- list-nodes,
- setup-ssh,
- update-node-ssh,
- version

Print-Certificate

Prints legible and detailed information of one or multiple SSL certificate and key entries.

Synopsis

```
asadmin [asadmin-options] print-certificate [--help]
[--certificatealias certificatealias]
[--providerclass providerclass]
[file]
```

Description

The `print-certificate` subcommand is useful to print legible and detailed information of one or multiple SSL certificate and key entries.

Since the output generated by the standard keytool utility is not standardized (especially distinguished names since they usually they use RFC 1779), it is recommended to use this command instead.

See printing-certificate-data for more information.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--certificatealias`: If the file is a keystore, this is an alias used to access the certificate. **`--providerclass`**: The name of a custom `java.security.Provider` implementation class to be preferred in this command execution

Operands

file: File containing the certificate

Examples

Example 1 Printing a DER certificate with one entry

This example prints a DER certificate with one entry.

```
asadmin print-certificate ./certificate.der

Found Certificate:
Subject:    UID=LDAP-Test,EMAILADDRESS=nobody@nowhere.space,CN
=PrintCertificateCommandTest,OU=Test Test\, Test,O=Payara Foundation,L=Pilsen,C=CZ
Validity:   Thu Aug 01 02:00:00 CEST 2019 - Fri Aug 02 02:00:00 CEST 2019
S/N:        1
Version:    3
Issuer:     UID=LDAP-Test,EMAILADDRESS=nobody@nowhere.space,CN
=PrintCertificateCommandTest,OU=Test Test\, Test,O=Payara Foundation,L=Pilsen,C=CZ
Public Key: RSA, 2048 bits
Sign. Alg.: SHA256withRSA (OID: 1.2.840.113549.1.1.11)
Command print-certificate executed successfully.
```

Example 2 Printing a specific certificate using a P12 format keystore

This example prints a specific certificate using a P12 format keystore.

```
asadmin> print-certificate --certificatealias s1as ./keystore.p12

Keystore Password>
Found Certificate:
Subject:    CN=localhost,OU=Payara,O=Payara Foundation,L=Great Malvern,ST
=Worcestershire,C=UK
Validity:   Tue Aug 06 14:06:14 CEST 2019 - Fri Aug 03 14:06:14 CEST 2029
S/N:        886895448
Version:    3
```

```
Issuer:     CN=localhost,OU=Payara,O=Payara Foundation,L=Great Malvern,ST
=Worcestershire,C=UK
Public Key: RSA, 2048 bits
Sign. Alg.: SHA256withRSA (OID: 1.2.840.113549.1.1.11)
Command print-certificate executed successfully.
```

Example 3 Plugging an alternative provider

This example plugins a different `java.security.Provider` implementation to get different outputs when running the `print-certificate` command. In this example, we use the **BouncyCastle** library, which leads to one significant difference: A highlighted, uppercase signature's algorithm name.

```
#Copy the library to the asadmin lib directory first
cp .../bcprov-jdk15on-1.62.jar .../payara7/glassfish/lib/asadmin/

#Then run the command
asadmin> print-certificate --providerclass
org.bouncycastle.jce.provider.BouncyCastleProvider ./certificate.der

Found Certificate:
Subject:    UID=LDAP-Test,EMAILADDRESS=nobody@nowhere.space,CN
=PrintCertificateCommandTest,OU=Test Test\, Test,O=Payara Foundation,L=Pilsen,C=CZ
Validity:   Thu Aug 01 02:00:00 CEST 2019 - Fri Aug 02 02:00:00 CEST 2019
S/N:        1
Version:    3
Issuer:     UID=LDAP-Test,EMAILADDRESS=nobody@nowhere.space,CN
=PrintCertificateCommandTest,OU=Test Test\, Test,O=Payara Foundation,L=Pilsen,C=CZ
Public Key: RSA, 2048 bits
Sign. Alg.: SHA256WITHRSA (OID: 1.2.840.113549.1.1.11)
Command print-certificate executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- printing-certificate-data

Purge-Jbatch-Repository

Purges the current Jakarta Batch repository.

Synopsis

```
asadmin [asadmin-options] purge-jbatch-repository [--help]
[app-tag]
```

Description

The `purge-jbatch-repository` subcommand will purge the current Jakarta Batch repository of all jobs scheduled for the referenced application's tag.

NOTE	The command will remove all current jobs created by the referenced application whether these jobs are running or not.

TIP	If you are having trouble finding the application's tag for your application, you can get it programmatically on a Java EE application by using the following code:

```
import com.ibm.jbatch.container.servicesmanager.ServicesManager;
import com.ibm.jbatch.container.servicesmanager.ServicesManagerImpl;
import com.ibm.jbatch.container.services.IPersistenceManagerService;
...

JobOperator jobOperator = getJobOperator();
Long executionId = jobOperator.start("job-name", new Properties());

ServicesManager manager = ServicesManagerImpl.getInstance();
IPersistenceManagerService ps = manager.getPersistenceManagerService();
String applicationTag = ps.getJobCurrentTag(jobid);
```

If using Maven, you'll have to add a dependency to IBM's **JBatch Container Runtime** to your project to access the classes referenced above:

```
<dependency>
    <groupId>com.ibm.jbatch</groupId>
    <artifactId>com.ibm.jbatch.container</artifactId>
    <version>2.1.1</version>
    <scope>provided</scope>
</dependency>
```

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand.

Operands

app-tag: The application tag whose Jakarta Batch jobs will be purged from the

repository.

+ The application's tag is structured like this:

+

```
<Configuration Identifier>:<Application Name>__/<Application Name>
```

+ Where **Configuration Identifier** is the identifier of the configuration associated to the instance or cluster where the application is deployed and **Application Name** is the name of the deployed application.

+ See the description of the command on how to acquire the application's tag in a programmatic manner.

Examples

To remove all current jobs for an application named *jbatch-test* deployed on the DAS (`server-config`) then run the following command:

```
asadmin> purge-jbatch-repository "server-config:jbatch-test__/jbatch-test"
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- list-batch-jobs,
- list-batch-job-executions,
- list-batch-job-steps,
- list-batch-runtime-configuration

Recover-Transactions

Manually recovers pending transactions.

Synopsis

```
asadmin [asadmin-options] recover-transactions [--help]
[--transactionlogdir transaction_log_dir]
[--target target_server_name] server_name
```

Description

The `recover-transactions` subcommand manually recovers pending transactions.

For an installation of multiple server instances, you can run the `recover-transactions` subcommand from a surviving server instance to recover transactions after a server failure. To use this subcommand in this way, the following conditions must be met:

- Delegated transaction recovery is disabled.
- Transaction logs are stored on a shared file system or in a database that is accessible to all server instances.

For a stand-alone server, do not use this subcommand to recover transactions after a server failure. For a stand-alone server, the `recover-transactions` subcommand can recover transactions only when a resource fails, but the server is still running. If a stand-alone server fails, only the full startup recovery process can recover transactions that were pending when the server failed.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--transactionlogdir`**: The location of the transaction logs for a server for which transaction recovery is requested. This option applies only if transaction logs are stored on a shared file system. **`--target`**: The target server that performs the recovery for the server that is specified by the server_name operand. The target server should be running.

Operands

server_name: For a stand-alone server, the value of this operand is typically `server`. Transactions are recovered only if a resource fails, but the server is still running.
For an installation of multiple server instances, the value of this operand is the name of the server for which the recovery is required. The in-flight transactions on this server will be recovered. If this server is running, recovery is performed by the same server. In this situation, the `--transactionlogdir` and `--target` options should be omitted. If the server is not running, the `--target` option is required, and the `--transactionlogdir` option is also required if transaction logs are stored on a shared file system.

Examples

Example 1 Recovering transactions on a running server

```
% asadmin recover-transactions server1
Transaction recovered.
```

Example 2 Recovering transactions for a server that is not running

```
% asadmin recover-transactions --transactionlogdir /logs/tx --target server1
server2
Transaction recovered.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- freeze-transaction-service,
- rollback-transaction,
- unfreeze-transaction-service
- administering-transactions" in Payara Server General Administration section,
- Transactions

Redeploy

Redeploys the specified component.

Synopsis

```
asadmin [asadmin-options] redeploy [--help]
--name component_name
[--upload={true|false}]
[--retrieve local_dirpath]
[--dbvendorname dbvendorname]
[--deploymentplan deployment_plan]
[--altdd alternate_deploymentdescriptor]
[--runtimealtdd runtime_alternate_deploymentdescriptor]
[--deploymentorder deployment_order]
[--enabled={true|false}]
[--generatermistubs={false|true}]
[--contextroot context_root]
[--precompilejsp={true|false}]
[--verify={false|true}]
[--virtualservers virtual_servers]
[--availabilityenabled={false|true}]
[--asyncreplication={true|false}]
[--lbenabled={true|false}]
[--keepstate={false|true}]
[--libraries jar_file[,jar_file]*]
[--target target]
[--type pkg-type]
[--properties(name=value)[:name=value]*]
[--warlibs={false|true}]
[file_archive|filepath]
```

Description

The `redeploy` subcommand redeploys an enterprise application, web application, module based on the Enterprise JavaBeans (EJB) specification (EJB module), connector module, or application client module that is already deployed or already exists. The `redeploy` subcommand preserves the settings and other options with which the application was originally deployed. The application must already be deployed. Otherwise, an error occurs.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--virtualservers`**: One or more virtual server IDs. Multiple IDs are

separated by commas. `--contextroot`: Valid only if the archive is a web module. It is ignored for other archive types; it will be the value specified by default-context-path in web.xml, if specified; defaults to filename without extension. `--precompilejsp`: By default this option does not allow the JSP to be precompiled during deployment. Instead, JSPs are compiled during runtime. Default is `false`. `--verify`: If set to true and the required verifier packages are installed from the Update Tool, the syntax and semantics of the deployment descriptor is verified. Default is `false`. `--name`: Name of the deployable component.

The name can include an optional version identifier, which follows the name and is separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. For more information about module and application versions, see module-and-application-versions in Payara Server Application Deployment section. `--upload`: Specifies whether the subcommand uploads the file to the DAS. In most situations, this option can be omitted.

Valid values are as follows:

`false`;; The subcommand does not upload the file and attempts to access the file through the specified file name. If the DAS cannot access the file, the subcommand fails.

For example, the DAS might be running as a different user than the administration user and does not have read access to the file. In this situation, the subcommand fails if the `--upload` option is `false`. `true`;; The subcommand uploads the file to the DAS over the network connection.

The default value depends on whether the DAS is on the host where the subcommand is run or is on a remote host.

* If the DAS is on the host where the subcommand is run, the default is `false`. * If the DAS is on a remote host, the default is `true`.

If a directory filepath is specified, this option is ignored. `--retrieve`: Retrieves the client stub JAR file from the server machine to the local directory. `--dbvendorname`: Specifies the name of the database vendor for which tables are created. Supported values include `h2`, `db2`, `mssql`, `oracle`, `derby`, `javadb`, `postgresql`, and `sybase`,

case-insensitive. If not specified, the value of the `database-vendor-name` attribute in `glassfish-ejb-jar.xml` is used. If the connection cannot be established, or if the value is not valid, SQL-92 compliance is presumed. **`--deploymentplan`**: Deploys the deployment plan, which is a JAR file that contains Payara Server descriptors. Specify this option when deploying a pure EAR file. A pure EAR file is an EAR without Payara Server descriptors. **`--altdd`**: Deploys the application using a Jakarta EE standard deployment descriptor that resides outside of the application archive. Specify an absolute path or a relative path to the alternate deployment descriptor file. The alternate deployment descriptor overrides the top-level deployment descriptor packaged in the archive. For example, for an EAR, the `--altdd` option overrides `application.xml`. For a standalone module, the `--altdd` option overrides the top-level module descriptor such as `web.xml`. -**`-runtimealtdd`**: Deploys the application using a Payara Server runtime deployment descriptor that resides outside of the application archive. Specify an absolute path or a relative path to the alternate deployment descriptor file. The alternate deployment descriptor overrides the top-level deployment descriptor packaged in the archive. For example, for an EAR, the `--runtimealtdd` option overrides `payara-application.xml`. For a standalone module, the `--runtimealtdd` option overrides the top-level module descriptor such as `glassfish-web.xml` (deprecated) or `payara-web.xml`. Applies to Payara Server deployment descriptors only (`glassfish-`**`.xml`**`/payara-.xml`); the name of the alternate deployment descriptor file must begin with `glassfish-` (or `payara-` in specific cases). Does not apply to `sun-`**`.xml`**` deployment descriptors, which are deprecated. *` `--deploymentorder`: Specifies the deployment order of the application. This is useful if the application has dependencies and must be loaded in a certain order at server startup. The deployment order is specified as an integer. The default value is 100. Applications with lower numbers are loaded before applications with higher numbers. For example, an application with a deployment order of 102 is loaded before an application with a deployment order of 110. If a deployment order is not specified, the default value of 100 is assigned. If two applications have the same deployment order, the first application to be deployed is the first application to be loaded at server startup. The deployment order is typically specified when the application is first deployed but

can also be specified or changed after initial deployment using the `set` subcommand. You can view the deployment order of an application using the `get` subcommand. **`--enabled`**: Allows users to access the application. If set to `false`, users will not be able to access the application. This option enables the application on the specified target instance or cluster. If you deploy to the target `domain`, this option is ignored, since deploying to the domain doesn't deploy to a specific instance or cluster. The default is `true`. **`--generatermistubs`**: If set to `true`, static RMI-IIOP stubs are generated and put into the `client.jar`. If set to `false`, the stubs are not generated. Default is `false`. **`--availabilityenabled`**: This option controls whether high-availability is enabled for web sessions and for stateful session bean (SFSB) checkpointing and potentially passivation. If set to false (default) all web session saving and SFSB checkpointing is disabled for the specified application, web application, or EJB module. If set to true, the specified application or module is enabled for high-availability. Set this option to true only if high availability is configured and enabled at higher levels, such as the server and container levels. **`--asyncreplication`**: This option controls whether web session and SFSB states for which high availability is enabled are first buffered and then replicated using a separate asynchronous thread. If set to `true` (default), performance is improved but availability is reduced. If the instance where states are buffered but not yet replicated fails, the states are lost. If set to `false`, performance is reduced but availability is guaranteed. States are not buffered but immediately transmitted to other instances in the cluster.

`--lbenabled`: This option controls whether the deployed application is available for load balancing. The default is `true`. **`--keepstate`**: This option controls whether web sessions, SFSB instances, and persistently created EJB timers are retained between redeployments.
The default is false. This option is supported only on the default server instance, named `server`. It is not supported and ignored for any other target.
Some changes to an application between redeployments prevent this feature from working properly. For example, do not change the set of instance variables in the SFSB bean class.

For web applications, this feature is applicable only if in the `payara-web.xml` file the `persistence-type` attribute of the `session-manager` element is `file`.
For stateful session bean instances, the persistence type without high availability is set in the server (the `sfsb-persistence-type` attribute) and must be set to `file`, which is the default and recommended value.
If any active web session, SFSB instance, or EJB timer fails to be preserved or restored, none of these will be available when the redeployment is complete. However, the redeployment continues and a warning is logged.
To preserve active state data, Payara Server serializes the data and saves it in memory. To restore the data, the class loader of the newly redeployed application deserializes the data that was previously saved. **`--warlibs`**: If set to true, enables an application to make use of shared libraries installed to `domain_dir/lib/warlibs`. Default is false. This parameter takes precedence over enabling warlibs in --properties. **`--libraries`**: A comma-separated list of library JAR files. Specify the library JAR files by their relative or absolute paths. Specify relative paths relative to domain-dir`/lib/applibs`. The libraries are made available to the application in the order specified. **`--target`**: Specifies the target to which you are deploying. Valid values are:
`server`;; Deploys the component to the default server instance `server` and is the default value. `domain`;; Deploys the component to the domain. If `domain` is the target for an initial deployment, the application is deployed to the domain, but no server instances or clusters reference the application. If `domain` is the target for a redeployment, and dynamic reconfiguration is enabled for the clusters or server instances that reference the application, the referencing clusters or server instances automatically get the new version of the application. If redeploying, and dynamic configuration is disabled, the referencing clusters or server instances do not get the new version of the application until the clustered or standalone server instances are restarted. cluster_name;; Deploys the component to every server instance in the cluster. instance_name;; Deploys the component to a particular stand-alone server instance. **`--type`**: The packaging archive type of the component that is being deployed. Possible values are as follows:
`car`;; The component is packaged as a CAR file. `ear`;; The component is packaged as

an EAR file. `ejb`;; The component is an EJB packaged as a JAR file. `osgi`;; The component is packaged as an OSGi bundle. `rar`;; The component is packaged as a RAR file. `war`;; The component is packaged as a WAR file. **`--properties`** or **`--property`**: Optional keyword-value pairs that specify additional properties for the deployment. The available properties are determined by the implementation of the component that is being deployed or redeployed. The `--properties` option and the `--property` option are equivalent. You can use either option regardless of the number of properties that you specify.

You can specify the following properties for a deployment:

`jar-signing-alias`;; Specifies the alias for the security certificate with which the application client container JAR file is signed. Java Web Start will not run code that requires elevated permissions unless it resides in a JAR file signed with a certificate that the user's system trusts. For your convenience, Payara Server signs the JAR file automatically using the certificate with this alias from the domain's keystore. Java Web Start then asks the user whether to trust the code and displays the Payara Server certificate information. To sign this JAR file with a different certificate, add the certificate to the domain keystore, then use this property. For example, you can use a certificate from a trusted authority, which avoids the Java Web Start prompt, or from your own company, which users know they can trust. Default is `s1as`, the alias for the self-signed certificate created for every domain. `java-web-start-enabled`;; Specifies whether Java Web Start access is permitted for an application client module. Default is true. `compatibility`;; Specifies the Payara Server release with which to be backward compatible in terms of JAR visibility requirements for applications. The only allowed value is `v2`, which refers to Sun Java System Application Server version 2 or Sun Java System Application Server version 9.1 or 9.1.1. Beginning in Java EE 6, the Java EE platform specification imposed stricter requirements than Java EE 5 did on which JAR files can be visible to various modules within an EAR file. In particular, application clients must not have access to EJB JAR files or other JAR files in the EAR file unless references use the standard Java SE mechanisms (extensions, for example) or the Jakarta EE library-directory mechanism. Setting this property to `v2` removes these restrictions. `keepSessions={false|true}`;; Superseded by the `--keepstate`

option.

This property can by used to specify whether active sessions of the application that is being redeployed are preserved and then restored when the redeployment is complete. Applies to HTTP sessions in a web **container. Default is `false`.**: `false`: Active sessions of the application are not preserved and restored (default). `true`: Active sessions of the application are preserved and restored.

If any active session of the application fails to be preserved or restored, none of the sessions will be available when the redeployment is complete. However, the redeployment continues and a warning is logged.

To preserve active sessions, Payara Server serializes the sessions and saves them in memory. To restore the sessions, the class loader of the newly redeployed application deserializes any sessions that were previously saved.

`preserveAppScopedResources`;; If set to `true`, preserves any application-scoped resources and restores them during redeployment. Default is `false`.

`warlibs={false|true}`

(This property will be overriden by `--warlibs`) If set to true, enables an application to make use of shared libraries installed to `domain_dir/lib/warlibs`. Default is `false`.

Libraries stored in this directory allows a developer to create and deploy skinny WARs, as the libraries stored in this directory are subject to additional annotation and extension scanning & processing for CDI, EJB, Servlet, and more. Unlike common and application libraries (those stored in `domain_dir/lib` and `domain_dir/lib/applibs` respectively), warlibs are cached and don't dynamically update if the files within the directory are modified.

```
Other available properties are determined by the implementation of the
component that is being redeployed. +
For components packaged as OSGi bundles (`--type=osgi`), the `deploy`
subcommand accepts properties arguments to wrap a WAR file as a WAB
(Web Application Bundle) at the time of deployment. The subcommand looks for
a key named `UriScheme` and, if present, uses the key as a
URL stream handler to decorate the input stream. Other properties are used in
the decoration process. For example, the Payara Server
OSGi web container registers a URL stream handler named `webbundle`, which is
used to wrap a plain WAR file as a WAB. For more information
about usage, see the related example in the xref:deploy.adoc#deploy[`deploy`]
```

```
help page.
```

Operands

file_archive|filepath: The path to the archive that contains the application that is being redeployed. This path can be a relative path or an absolute path.
The archive can be in either of the following formats:
* An archive file, for example, `/export/JEE_apps/hello.war`.
If the `--upload` option is set to `true`, this is the path to the deployable file on the local client machine. If the `--upload` option is set to `false`, this is the absolute path to the file on the server machine. * A directory that contains the exploded format of the deployable archive. This is the absolute path to the directory on the server machine.
If you specify a directory, the `--upload` option is ignored.
Whether this operand is required depends on how the application was originally deployed:
* If the application was originally deployed from a file, the archive-path operand is required. The operand must specify an archive file. * If the application was originally deployed from a directory, the archive-path operand is optional.
The operand can specify a directory or an archive file.

Examples

Example 1 Redeploying a Web Application From a File

This example redeploys the web application `hello` from the `hello.war` file in the current working directory. The application was originally deployed from a file. Active sessions of the application are to be preserved and then restored when the redeployment is complete.

```
asadmin> redeploy --name hello --properties keepSessions=true hello.war
Application deployed successfully with name hello.
Command redeploy executed successfully
```

Example 2 Redeploying a Web Application From a Directory

This example redeploys the web application `hellodir`. The application was originally deployed from a directory.

```
asadmin> redeploy --name hellodir
Application deployed successfully with name hellodir.
Command redeploy executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- deploy,
- get,
- undeploy

Remove-Expired-Certificates

Removes all expired certificates from the target instance or listener's keystores and truststores.

Synopsis

```
asadmin remove-expired-certificates [--help]
[--reload={false|true}]
[--domainName=name]
[--domainDir=directory]
[--node=nodeName]
[--nodeDir=directory]
[--listener=listenername]
[--target=instanceName]
```

Description

This command removes all expired certificates from the target instance or listener's key and trust stores.

If the instance or listener is configured to use the default trust store, the command will instead synchronize the instance with the DAS (under the assumption the certificate has been removed from the default trust store of the DAS), since any certificates removed from the instance stores would be lost upon its next synchronisation.

Options

`--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are removing expired certificates. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. `--reload`: Whether the HTTP listeners should be reloaded. Defaults to `false` `--listener`: The name of the HTTP or IIOP listener to add the certificate to. `--domainname`: The name of the domain where the target instance exists.

+ Defaults to `domain1` or the existing domain if only one exists. `--domainDir`: The path to the directory containing the target domain.

+ Defaults to `as-install/glassfish/domains` `--node`: The name of the node where the target instance exists.

+ Defaults to `localhost-$domainname` `--nodeDir`: The path to the directory containing the target node.

+ Defaults to `as-install/glassfish/nodes`

Examples

Example 1 Removes all expired certificates in the `production` domain

```
asadmin remove-expired-certificates --domainName --listener http-listener-2
--reload=true
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- "SSL Certificate Management",
- "`remove-from-keystore`",
- "`remove-from-truststore`"

Remove-From-Keystore

Removes a key/certificate entry from the specified server keystore.

Synopsis

```
asadmin remove-from-keystore [--help]
[--reload={false|true}]
[--domainName=name]
[--domainDir=directory]
[--node=nodeName]
[--nodeDir=directory]
[--listener=listenername]
[--target=instanceName]
[alias]
```

Description

This command removes a private key and certificate entry from the target instance or listener's keystore matching the provided alias.

If the instance or listener is configured to use the default key store, the command will instead synchronize the instance with the DAS (under the assumption the certificate has been removed from the default key store of the DAS), since any certificates removed from the instance stores would be lost upon its next synchronisation.

Options

`--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are removing from its keystore. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. `--reload`: Whether the HTTP listeners should be reloaded. Defaults to `false` `--listener`: The name of the HTTP or IIOP listener to add the certificate to. `--domainname`: The name of the domain where the target instance exists.

+ Defaults to `domain1` or the existing domain if only one exists. `--domainDir`: The path to the directory containing the target domain.

+ Defaults to `as-install/glassfish/domains` `--node`: The name of the node where the target instance exists.

+ Defaults to `localhost-$domainname` **`--nodeDir`**: The path to the directory containing the target node.

+ Defaults to `as-install/glassfish/nodes`

Operands

alias: The alias name of the private key/certificate entry to remove.

Examples

Example 1 Remove the `mycert` entry from the `production` domain's keystore

```
asadmin remove-from-keystore --domainName production --listener http-listener-2
mycert
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- "SSL Certificate Management",
- "`remove-expired-certificates`",
- "`remove-from-truststore`"

Remove-From-Truststore

Removes a certificate entry from the specified server truststore.

Synopsis

```
asadmin remove-from-truststore [--help]
[--reload={false|true}]
[--domainName=name]
[--domainDir=directory]
[--node=nodeName]
[--nodeDir=directory]
[--listener=listenername]
[--target=instanceName]
[alias]
```

Description

This command removes a certificate from the target instance or listener's truststore

matching the provided alias.

If the instance or listener is configured to use the default trust store, the command will instead synchronize the instance with the DAS (under the assumption the certificate has been removed from the default trust store of the DAS), since any certificates removed from the instance stores would be lost upon its next synchronisation.

Options

`--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are removing from its truststore. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. `--reload`: Whether the HTTP listeners should be reloaded. Defaults to `false` `--listener`: The name of the HTTP or IIOP listener to add the certificate to. `--domainname`: The name of the domain where the target instance exists.

+ Defaults to `domain1` or the existing domain if only one exists. `--domainDir`: The path to the directory containing the target domain.

+ Defaults to `as-install/glassfish/domains` `--node`: The name of the node where the target instance exists.

+ Defaults to `localhost-$domainname` `--nodeDir`: The path to the directory containing the target node.

+ Defaults to `as-install/glassfish/nodes`

Operands

alias: The alias name of the certificate entry to remove.

Examples

Example 1 Remove the `mycert` entry from the `production` domain's truststore

```
asadmin remove-from-truststore --domainName production --listener http-listener-2
mycert
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- "SSL Certificate Management",
- "`remove-expired-certificates`",
- "`remove-from-keystore`"

Remove-Instance-From-Deployment-Group

Removes a standalone instance from an existing deployment group.

Synopsis

```
asadmin [asadmin-options] remove-instance-from-deployment-group [--help]
[--instance instanceName]
[--deploymentgroup deploymentGroupName]
```

Description

The `remove-instance-from-deployment-group` subcommand will remove the referenced instance from the referenced deployment group.

If the referenced instance doesn't belong to the deployment group the command will exit with an error.

IMPORTANT	Removing an instance from a deployment group does not delete it.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--instance`: The name of the standalone instance that will be removed from the

deployment group. The command will fail if the name doesn't match to an existing standalone instance that belongs to the deployment group.

`--deploymentGroup`: The name of the deployment group from which the standalone instance will be removed. The command will fail if the name doesn't match to an existing deployment group.

Examples

Example 1 Removes an instance from a Deployment Group

This example removes the instance named `test-instance-1` from the deployment group named `test-dgroup`:

```
asadmin remove-instance-from-deployment-group --instance=test-instance-1
--deploymentGroup=test-dgroup

Command remove-instance-from-deployment-group executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-deployment-group,
- delete-deployment-group,
- list-deployment-groups

Remove-Library

Removes one or more library JAR files from Payara Server.

Synopsis

```
asadmin [asadmin-options] remove-library [--help]
[--type={common|ext|app}]
library-name [library-name ... ]
```

Description

The `remove-library` subcommand removes one or more library JAR files from Payara Server.

The `--type` option specifies the library type and the Payara Server directory from which the library is removed.

The library-name operand is the name of the JAR file that contains the library to be removed. To specify multiple libraries, specify multiple names separated by spaces.

NOTE	The library archive file is removed from the DAS. For common and extension libraries, you must restart the DAS so the library removals are picked up by the server runtime. To remove the libraries from other server instances, synchronize the instances with the DAS by restarting them.

This command is not supported on the Windows operating system.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--type`**: Specifies the library type and the Payara Server directory from which the library is removed. Valid values are as follows:
`common`;; Removes the library files from the Common class loader directory, domain-dir`/lib`. This is the default. `ext`;; Removes the library files from the Java optional package directory, domain-dir`/lib/ext`. `app`;; Removes the library files from the application-specific class loader directory, `domain-dir/lib/applibs`. `war`;; Removes the library files from the war-packaged class loader directory, `domain-dir/lib/warlibs`.
For more information about these directories, see classloading in the Payara Server Application Development section.

Operands

library-name: The names of the JAR files that contain the libraries that are to be removed.

Examples

Example 1 Removing Libraries

This example removes the library in the archive file `mylib.jar` from the application-specific class loader directory on the default server instance.

```
asadmin> remove-library --type app mylib.jar
Command remove-library executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- add-library,
- list-libraries,
- classloading in the Payara Server Application Development section.

Renew-Self-Signed-Certificates

Renews all self-signed certificates in the specified server's keystore.

Synopsis

```
asadmin renew-self-signed-certificates [--help]
[--reload={false|true}]
[--domainName=name]
[--domainDir=directory]
[--node=nodeName]
[--nodeDir=directory]
[--listener=listenername]
[--target=instanceName]
```

Description

The `renew-self-signed-certificates` subcommand renews all self-signed

certificates in a keystore using the old distinguished name and alias.

These certificates are then placed in the key and trust stores of the target instance or listener, replacing the old ones.

Options

`--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are renewing its self-signed certificates. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. `--reload`: Whether the HTTP listeners should be reloaded. Defaults to `false` `--listener`: The name of the HTTP or IIOP listener to add the certificate to. `--domainname`: The name of the domain where the target instance exists.

+ Defaults to `domain1` or the existing domain if only one exists. `--domainDir`: The path to the directory containing the target domain.

+ Defaults to `as-install/glassfish/domains` `--node`: The name of the node where the target instance exists.

+ Defaults to `localhost-$domainname` `--nodeDir`: The path to the directory containing the target node.

+ Defaults to `as-install/glassfish/nodes`

Examples

Example 1 Renews all self-signed certificates on the default domain

```
asadmin renew-self-signed-certificates
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- "SSL Certificate Management",
- "`generate-csr`",
- "`generate-self-signed-certificate`",
- "`add-to-keystore`",
- "`add-to-truststore`"

Resource

Provide connectivity to various types of EIS and other integrated systems.

Description

Payara Server provides support for JDBC, JMS, and JNDI resources.

See Also

- asadmin-1m

Restart-Cluster

Restarts all instances in a cluster.

Synopsis

```
asadmin [asadmin-options] restart-cluster [--help]
[--verbose=false]
[--rolling=true]
[--delay=0]
[--instanceTimeout=instanceTimeout]
[--timeout=timeout]
[clustername]
```

Description

The `restart-cluster` subcommand stops and starts a cluster by stopping and starting all Payara Server instances in the cluster that are already running.

This subcommand requires the secure shell (SSH) to be configured on the host where the domain administration server (DAS) is running and on all hosts where instances in the cluster reside.

NOTE	If all instances reside on the same host as the DAS, SSH is not required. You might require to restart a cluster in which instances reside on hosts where SSH is not configured that are remote from the DAS. In this situation, run the restart-local-instance subcommand for each instance from the host where the instance resides.

You may run this subcommand from any host that can contact the DAS.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--verbose`: Specifies whether additional status information is displayed when the cluster is started.

+ Valid values are as follows:

+ `true`;; Displays the command to start each instance in the cluster and whether the attempt to start each instance succeeded. `false`;; Displays no additional status information (default).

`--rolling`: Performs a rolling restart which each instance restarted in turn. The default value is `true`.

`--delay`: Delay in (ms) between each instance restart in a rolling restart. The default value is `0`.

`--instanceTimeout`: Specify a timeout value of instances. The default value is `600`.

`--timeout`: Specify a timeout value of the command. The default value is `600`.

Operands

cluster-name: The name of the cluster to restart.

Examples

Example 1 Restarting a Cluster

This example restarts the cluster `c1`.

```
asadmin> restart-cluster c1
Command restart-cluster executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-cluster,
- delete-cluster,
- list-clusters,
- setup-ssh,
- start-local-instance,
- start-cluster,
- stop-cluster

Restart-Deployment-Group

Restarts all the instances in a Payara Server deployment group.

Synopsis

```
asadmin [asadmin-options] restart-deployment-group [--help]
[--instanceTimeout timeout]
[--timeout timeout]
[--verbose={false|true}]
[--rolling={false|true}]
[--delay delay]
name
```

Description

The `restart-deployment-group` subcommand will restart all instances that belong to a Payara Server deployment group. The command will fail if the deployment group doesn't contain any instances in it. To this effect, the command will:

1. Stop any running instances first and wait until all instances are stopped.
2. Start all instances that belong to the deployment group.

You can configure the command to execute a "rolling restart", that is, instead of stopping all instances first and then starting them, stop and start each instance in order by using the `--rolling` option.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--verbose`: **`-v`**: Whether to start the instances in verbose mode. Defaults to `false`.

`--timeout`: Specifies how long to take for command execution to be completed in seconds. If the execution takes longer than this amount then this command will fail.

+ The default value is `600`.

`--instanceTimeout`: Specifies how long to take for each instance to start/stop in seconds. If each operation takes longer than this amount then this command will fail.

+ The default value is `600`.

`--rolling`: Whether to execute a "rolling restart", that is, stop and start each instance in order to prevent a full shutdown of an application. Defaults to `false`. **`--delay`**: Only applies when `--rolling` is set to `true`. Sets a delay wait time in milliseconds between each instance restart. Defaults to `5000`.

Operands

name: The name of the deployment group. If there are no deployment groups that match the name the command will exit with an error.

Examples

Example 1 Restart a Deployment Group

This example restarts a deployment group that is named `test-dgroup`:

```
asadmin restart-deployment-group test-dgroup
Command restart-deployment-group executed successfully.
```

Example 2 Execute a rolling restart of a Deployment Group

This example restarts a deployment group that is named `test-dgroup` in rolling mode using a delay of 30 seconds:

```
asadmin restart-deployment-group --rolling=true --delay=30000 test-dgroup
Command restart-deployment-group executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-deployment-group,
- start-instance,
- stop-instance,
- list-deployment-groups,
- start-deployment-group,
- stop-deployment-group

Restart-Domain

Restarts the Domain Administration Server(DAS) of the specified domain.

Synopsis

```
asadmin [asadmin-options] restart-domain [--help]
[--debug ={true|false}]
[--domaindir domaindir]
[--timeout timeout]
[--force={true|false}] [--kill={false|true}]
[domain-name]
```

Description

The `restart-domain` subcommand stops and then restarts the domain administration server (DAS) of the specified domain. If a domain is not specified, the default domain is assumed. If the domains directory contains two or more domains, the domain-name operand must be specified. If the DAS is not already running, the subcommand attempts to start it.

The `restart-domain` subcommand does not exit until the subcommand has verified that the domain has been stopped and restarted.

This subcommand is supported in local or remote mode. If you specify a host name, the subcommand assumes you are operating in remote mode, which means you must correctly authenticate to the remote server. In local mode, you normally do not need to authenticate to the server as long as you are running the subcommand as the same user who started the server.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--debug`**: Specifies whether the domain is restarted with Java Platform Debugger Architecture (JPDA) (https://docs.oracle.com/javase/8/docs/technotes/guides/jpda/index.html) debugging enabled.
Possible values are as follows:
`true`;; The domain is restarted with JPDA debugging enabled and the port number for JPDA debugging is displayed. `false`;; The domain is restarted with JPDA debugging

disabled (default).
The default is the current setting of this option for the domain that is being restarted.
`--domaindir`: The domain root directory, which contains the directory of the domain that is to be restarted. If specified, the path must be accessible in the file system. The default location of the domain root directory is as-install`/domains`. `--force`: Specifies whether the domain is forcibly stopped immediately before it is restarted.
Possible values are as follows:
`true`;; The domain is forcibly stopped immediately (default). `false`;; The subcommand waits until all threads that are associated with the domain are exited before stopping the domain. `--kill`: Specifies whether the domain is killed before it is restarted by using functionality of the operating system to terminate the domain process.
Possible values are as follows:
`false`;; The domain is not killed. The subcommand uses functionality of the Java platform to terminate the domain process (default). `true`;; The domain is killed. The subcommand uses functionality of the operating system to terminate the domain process. `--timeout`: Specifies how long to take for the domains to restart in seconds. If restarting the domains takes longer than this amount then this command will fail.

+ The default value is `600`.

Operands

domain-name: The name of the domain you want to restart. Default is the name specified during installation, usually `domain1`.

Examples

Example 1 Restarting a Domain

This example restarts `mydomain4` in the default domains directory.

```
asadmin> restart-domain mydomain4
Successfully restarted the domain
Command restart-domain executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-domain,
- delete-domain,
- list-domains,
- restart-domain,
- start-domain,
- start-domains,
- stop-domain,
- stop-domains,
- stop-all-domains`]
- Java Platform Debugger Architecture (JPDA) (`https://docs.oracle.com/javase/8/docs/technotes/guides/jpda/index.html/)

Restart-Domains

Restarts the Domain Administration Server(DAS) of the specified domains.

Synopsis

```
asadmin [asadmin-options] restart-domains [--help]
[--domainTimeout timeout]
[--timeout timeout]
[domain-name]
```

Description

The `restart-domains` subcommand can be used to restart the specified list of domains in the operand. This list must be specified as a comma separated string of values.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--timeout`: Specifies how long to take for the domains to restart in seconds. If restarting the domains takes longer than this amount then this command will fail.

+ The default value is `600`.

`--domainTimeout`: Specifies how long to take for each domain to restart in seconds. If restarting the domain takes longer than this amount then this command will fail.

+ The default value is `600`.

Operands

domain-names: The name of the domains you want to restart.

Examples

Example 1 Restarting Domains

This example restarts `domain1` and `testy` in the default domains directory.

```
asadmin restart-domains domain1,testy

Successfully restarted the domain
Restarted domain domain1

Successfully restarted the domain
Restarted domain testy

Command restart-domains executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-domains,

- restart-domain,
- start-domain,
- start-domains,
- stop-domain,
- stop-domains,
- stop-all-domains
- Java Platform Debugger Architecture (JPDA) (`https://docs.oracle.com/javase/8/docs/technotes/guides/jpda/index.html/)

Restart-Hazelcast

Restarts the Data Grid/Hazelcast in an specific target.

Synopsis

```
asadmin [asadmin-options] restart-hazelcast
[--help]
[--target target]
```

Description

This command can be used to restart the Data Grid/Hazelcast subsystem in the specified target.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are restarting its corresponding Hazelcast member. Valid values are: `server`;; Applies to the default server instance. This is the default value. instance_name;; Applies to a specified server instance.

Examples

Example 1 Restarts the Data Grid/Hazelcast

This example restarts the Data Grid/Hazelcast with `instance1` as its target.

```
asadmin> restart-hazelcast --target instance1
instance1:
Hazelcast Restarted

Command restart-hazelcast executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Domain Data Grid",
- "`list-hazelcast-cluster-members`",
- "`list-hazelcast-members`",
- "`list-cache-keys`",
- "`list-caches`",
- "`clear-cache`",
- "`set-hazelcast-configuration`",
- "`get-hazelcast-configuration`",
- "`generate-encryption-key`"

Restart-Http-Listeners

```
Restarts all HTTP network listener of the targeted instance(s).
```

Synopsis

```
asadmin [asadmin-options] restart-http-listeners
[--help]
[--all={true|false}]
[target]
```

Description

The `restart-http-listeners` subcommand restarts an HTTP network listener of the targeted instance(s).

IMPORTANT	This command only restarts all HTTP listeners except `admin-listener`.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: `-?`: Displays the help text for the subcommand.

`--all`: Applies the command to all instances. The default value is `false`.

Operands

target: Name of the target instance, deployment group or cluster to which restart the corresponding HTTP listeners.

Examples

Example 1 Restarting an HTTP Network Listener

The following command to restart all HTTP network listeners that belong to a particular instance named `i1`:

```
asadmin>restart-http-listeners i1
Command restart-http-listeners executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-ssl,
- create-network-listener,
- set-network-listener-configuration,
- create-virtual-server,

- create-http-listener,
- delete-http-listener,
- list-http-listeners,
- restart-http-listeners

Restart-Instance

Restarts a running Payara Server instance.

Synopsis

```
asadmin [asadmin-options] restart-instance [--help]
[--debug={false|true}] instance-name
```

Description

The `restart-instance` subcommand restarts a running Payara Server instance. This subcommand requires secure shell (SSH) to be configured on the machine where the domain administration server (DAS) is running and on the machine where the instance resides.

NOTE	SSH is not required if the instance resides on a node of type `CONFIG` that represents the local host. A node of type `CONFIG` is not enabled for communication over SSH.

You may run this subcommand from any machine that can contact the DAS.

The subcommand can restart any Payara Server instance, regardless of how the instance was created. For example, this subcommand can restart an instance that was created by using the create-local-instance subcommand.

When this subcommand restarts an instance, the DAS synchronizes the instance with changes since the last synchronization as follows:

- For the `config` directory, the DAS synchronizes the instance with all changes.
- For the `applications` directory and `docroot` directory, only a change to a top-level

subdirectory causes the DAS to synchronize all files under that subdirectory.
If a file below a top level subdirectory is changed without a change to a file in the top level subdirectory, full synchronization is required. In normal operation, files below the top level subdirectories of these directories are not changed. If an application is deployed and undeployed, full synchronization is not necessary to update the instance with the change.

If different synchronization behavior is required, the instance must be stopped and restarted by using following sequence of subcommands:

1. stop-instance
2. start-instance

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--debug`**: Specifies whether the instance is restarted with Java Platform Debugger Architecture (JPDA)(https://docs.oracle.com/javase/8/docs/technotes/guides/jpda/index.html/) debugging enabled.
Possible values are as follows:
`true`;; The instance is restarted with JPDA debugging enabled and the port number for JPDA debugging is displayed. `false`;; The instance is restarted with JPDA debugging disabled.
The default is the current setting of this option for the instance that is being restarted.

Operands

instance-name: The name of the Payara Server instance to restart. If the instance is not running, the subcommand displays a warning message and attempts to start the instance.

Examples

Example 1 Restarting a Payara Server Instance

This example restarts the Payara Server instance `pmdsa1`.

```
asadmin> restart-instance pmdsa1
Instance pmdsa1 was restarted.

Command restart-instance executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-instance,
- create-local-instance,
- delete-instance,
- delete-local-instance,
- restart-local-instance,
- setup-ssh,
- start-instance,
- start-local-instance,
- stop-instance,
- stop-local-instance
- Java Platform Debugger Architecture (JPDA) (http://www.oracle.com/technetwork/java/javase/tech/jpda-141715.html)

Restart-Local-Instance

Restarts a running Payara Server instance on the host where the subcommand is running.

Synopsis

```
asadmin [asadmin-options] restart-local-instance [--help]
```

```
[--nodedir nodedir] [--node node]
[--debug={false|true}]
[--force={true|false}] [--kill={false|true}]
[instance-name]
```

Description

The `restart-local-instance` subcommand restarts a Payara Server instance on the host where the subcommand is run. This subcommand does not require secure shell (SSH) to be configured. You must run this command from the host where the instance resides.

The subcommand can restart any Payara Server instance, regardless of how the instance was created. For example, this subcommand can restart an instance that was created by using the create-instance subcommand.

The `restart-local-instance` subcommand does not contact the domain administration server (DAS) to determine the node on which the instance resides. To determine the node on which the instance resides, the subcommand searches the directory that contains the node directories. If multiple node directories exist, the node must be specified as an option of the subcommand.

When this subcommand restarts an instance, the DAS synchronizes the instance with changes since the last synchronization as follows:

- For the `config` directory, the DAS synchronizes the instance with all changes.
- For the `applications` directory and `docroot` directory, only a change to a top-level subdirectory causes the DAS to synchronize all files under that subdirectory.
  If a file below a top level subdirectory is changed without a change to a file in the top level subdirectory, full synchronization is required. In normal operation, files below the top level subdirectories of these directories are not changed. If an application is deployed and undeployed, full synchronization is not necessary to update the instance with the change.

If different synchronization behavior is required, the instance must be stopped and restarted by using following sequence of subcommands:

1. stop-local-instance
2. start-local-instance

This subcommand is supported in local mode. However, to synchronize the instance with the DAS, this subcommand must be run in remote mode.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--nodedir`**: Specifies the directory that contains the instance's node directory. The instance's files are stored in the instance's node directory. The default is as-install`/nodes`. **`--node`**: Specifies the node on which the instance resides. This option may be omitted only if the directory that the `--nodedir` option specifies contains only one node directory. Otherwise, this option is required. **`--debug`**: Specifies whether the instance is restarted with Java Platform Debugger Architecture (JPDA) (http://www.oracle.com/technetwork/java/javase/tech/jpda-141715.html) debugging enabled.
Possible values are as follows:
`true`;; The instance is restarted with JPDA debugging enabled and the port number for JPDA debugging is displayed. `false`;; The instance is restarted with JPDA debugging disabled (default).
The default is the current setting of this option for the instance that is being restarted.
`--force`: Specifies whether the instance is forcibly stopped immediately before it is restarted.
Possible values are as follows:
`true`;; The instance is forcibly stopped immediately (default). `false`;; The subcommand waits until all threads that are associated with the instance are exited before stopping the instance. **`--kill`**: Specifies whether the instance is killed before it is restarted by using functionality of the operating system to terminate the instance process.
Possible values are as follows:
`false`;; The instance is not killed. The subcommand uses functionality of the Java

platform to terminate the instance process (default). `true`;; The instance is killed. The subcommand uses functionality of the operating system to terminate the instance process.

Operands

instance-name: The name of the Payara Server instance to restart. If the instance is not running, the subcommand displays a warning message and attempts to start the instance.

Examples

Example 1 Restarting an Instance Locally

This example restarts the instance `ymlsa1` in the domain `domain1` on the host where the subcommand is run.

```
asadmin> restart-local-instance --node localhost-domain1 ymlsa1
Command restart-local-instance executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-instance,
- create-local-instance,
- delete-instance,
- delete-local-instance,
- restart-instance,
- start-instance,
- start-local-instance,
- stop-instance,

- stop-local-instance
- Java Platform Debugger Architecture (JPDA) (http://www.oracle.com/technetwork/java/javase/tech/jpda-141715.html)

Restart-Monitoring

This command can be used to set restart/reset the monitoring levels of the server

Synopsis

```
asadmin [asadmin-options] restart-monitoring
[--help]
[--verbose={false|true}]
[--target target]
```

Description

This command can be used to restart the monitoring levels of all modules per the specified target.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the monitoring level of a server module. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--verbose`**: Enable verbose output.

Examples

Example 1 Restart monitoring in the DAS

```
asadmin restart-monitoring --target=server
```

This will generate an output similar to the following:

```
Restarted 3 modules
```

Example 2 Restart monitoring in the DAS using verbose mode

You can use the verbose mode to see which modules were affected and what monitoring level they are currently set to:

```
asadmin restart-monitoring --target=server --verbose
```

This will generate an output similar to the following:

```
Module                        Monitoring Level
cloud                         HIGH
cloud-elasticity              LOW
cloud-orchestrator            HIGH
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Monitoring Basics",
- "`set-monitoring-service-configuration`",
- "`enable-monitoring`",
- "`disable-monitoring`"

Restore-Domain

Restore files from backup.

Synopsis

```
asadmin [asadmin-options] restore-domain [--help]
[--long[={false|true}]]
[--filename backup-filename]
[--domaindir domain-root-dir]
[--backupdirbackup-directory]
[--backupconfigbackup-config-name]
[--force[={false|true}]]
[domain-name]
```

Description

This command restores files under the domain from a backup directory.

The `restore-domain` command is supported in local mode only.

WARNING	If a domain is restored on an already existing domain folder, all the files will be overwritten when the command is run.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**:

```
Displays the help text for the subcommand.
```

`--long`: **`-l`**: Displays detailed information about the restore operation.

+ The default value is `false`.

`--filename`: Specifies the name of the backup file to use as the source.

`--domaindir`: Specifies the domain root directory, the parent directory of the domain to restore.

+ The default value is `as-install/domains`.

`--backupdir`: Specifies the directory under which the backup file is stored.

+ The default value is `as-install/domains/domain-dir/backups`. If the domain is not in the default location, the location is `domain-dir/backups`.

`--backupconfig`: The name of the domain backup configuration in the backup directory under which the backup file is stored.

`--force`: Causes the restore operation to continue even when the name of the domain to restore does not match the name of the domain stored in the backup file.

+ The default value is `false`.

Operands

domain-name: Specifies the name of the domain to restore.

+ This operand is optional if only one domain exists in the Payara Server installation.

+ If the specified domain name does not match the domain name stored in the backup file, an error occurs unless the `--force` option is specified.

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- backup-domain,
- list-backups

Rollback-Transaction

Rolls back the named transaction.

Synopsis

```
asadmin [asadmin-options] rollback-transaction [--help]
[--target target]
transaction_id
```

Description

The `rollback-transaction` subcommand rolls back the named transaction.

Before you can roll back a transaction, you must do the following:

1. Enable monitoring using the `set` subcommand. For example:

```
asadmin> set clstr1-config.monitoring-service.module-monitoring-
levels.transaction-service=HIGH
```

2. Use the `freeze-transaction-service` subcommand to halt in-process transactions.
3. Look up the active transactions using the `get` subcommand with the `--monitor` option. For example:

```
asadmin> get --monitor inst1.server.transaction-service.activeids-current
```

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target on which you are rolling back the transactions. Valid values are `server` or any other clustered or stand-alone instance.

Operands

transaction_id: Identifier for the transaction to be rolled back.

Examples

Example 1 Using rollback-transaction command

```
% asadmin rollback-transaction 0000000000000001_00
Command rollback-transaction executed succeessfully
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- freeze-transaction-service,
- recover-transactions,
- unfreeze-transaction-service,

- "administering-transactions" in the Payara Server General Administration section
- "Transactions" in The Jakarta EE Specifications

Rotate-Log

Rotates the log file.

Synopsis

```
asadmin [asadmin-options] rotate-log [--help]
```

Description

The `rotate-log` subcommand rotates the server log by renaming the file with a timestamp name in the format `server.log_`date-and-time, and creating a new log file. Changes take effect dynamically, that is,server restart is not required.

The size of the log queue is configurable through the `logging.properties` file. Log rotation is based on file size or elapsed time since the last log rotation. In some circumstances, the queue might fill up, especially if the log level is set to `FINEST` and there is heavy activity on the server. In this case, the `rotate-log` subcommand can be used to rotate the server log immediately. This subcommand is also useful in creating scripts for rotating the log at convenient times.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: The server, cluster, or server instance for which logs will be rotated. If this option is omitted, logs are rotated for the default server.

Operands

target: Valid values are:
* server_name - Default target is `server`. If no target is specified then logs are rotated for the server. * cluster_name - The name of a target cluster. * instance_name - The name of a target server instance.

Examples

Example 1 Rotating the Server Log

This example rotates the server log.

```
asadmin> rotate-log
Command rotate-log executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- collect-log-files,
- list-log-attributes,
- list-log-levels,
- set-log-attributes,
- set-log-levels
- "administering-the-logging-service" in Payara Server General Administration section

Set-Admin-Audit-Configuration

Configures the Administrative Audit Service.

Synopsis

```
asadmin [asadmin-options] set-admin-audit-configuration [--help]
[--dynamic=false]
[--enabled={true|false}]
[--auditlevel=auditlevel]
[--target=server]
[--enablenotifiers=enablenotifiers]
[--disablenotifiers=disablenotifiers]
[--setnotifiers=setnotifiers]
```

Description

The `set-admin-audit-configuration` subcommand can be used to set the configuration settings of the Admin Audit Configuration.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **--dynamic**: Applies changes dynamically to the running system without the need for a restart, allowing for immediate effect of the audit configuration adjustments. **--enabled**: Turns the audit service on or off, controlling whether administrative actions are tracked and logged for security and compliance purposes. **--auditlevel**: Sets the auditing level. One of:

+ * MODIFIERS (default) * ACCESSORS * INTERNAL **--target**: Specifies the target for which to configure the audit settings, in this case, a server. **--enablenotifiers**: Enables selected notification services for audit events. **--disablenotifiers**: Disables selected notification services without affecting others. **--setnotifiers**: Configures notification services, specifying event types and settings.

Examples

Example 1

```
asadmin> set-admin-audit-configuration --dynamic --enabled --auditlevel=MODIFIERS
--target=server --enablenotifiers=email,alert --disablenotifiers=log --
setnotifiers="email:critical,alert:all"

Command set-admin-audit-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-admin-audit-configuration

Set-Amx-Enabled

Enables or disables the AMX monitoring subsystem.

Synopsis

```
asadmin [asadmin-options] set-amx-enabled
[--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
```

Description

This command can be used to configure the boot AMX at startup or independently of the JMX Monitoring service.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring AMX boot. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--enabled`**: Defaults to `false`. Enables or disables the monitoring service **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart.

Examples

Example 1 Enables the AMX subsystem immediately for the DAS

```
asadmin set-amx-enabled --enabled=true --dynamic=true --target=server
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Monitoring Basics"
- "`set-jmx-monitoring-configuration`"

Set-Asadmin-Recorder-Configuration

Configures the Asadmin Recorder service.

Synopsis

```
asadmin [asadmin-options] set-asadmin-recorder-configuration [--help]
[--enabled={true|false}]
[--outputlocation=outputlocation]
[--filtercommands={true|false}]
[--filteredcommands=filteredcommands]
[--prependenabled={true|false}]
[--prependedoptions=prependedoptions]
```

Description

The `set-asadmin-recorder-configuration` subcommand adjusts the settings of the Asadmin Recorder. When enabled with --enabled true, it starts recording all asadmin commands issued through the Admin Console.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **--enabled**: Activates or deactivates the Asadmin Recorder feature. **--outputLocation**: **-o**: The absolute file path for the asadmin commands to be written to. This defaults to a file called asadmin-commands.txt in the domain directory. **--filterCommands**: **-f**: Specifies whether the commands specified with the `--filteredcommands` option should be excluded from being recorded or not. **--filteredCommands**: A comma separated list of asadmin commands and regular expressions to exclude from being recorded.

Examples

Example 1

The following will enable the asadmin recorder, set the output location to `/home/user/asadmin-commands.txt`, and prevent the start-instance command from being recorded (in addition to the default configuration):

```
asadmin set-asadmin-recorder-configuration --enabled true --outputLocation
/home/user/asadmin-commands.txt --filterCommands true --filteredCommands
"version,_(.*),list(.*),get(.*),uptime,enable-asadmin-recorder,disable-asadmin-
```

```
recorder,set-asadmin-recorder-configuration,asadmin-recorder-enabled,start-
instance"

Command set-asadmin-recorder-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-asadmin-recorder-configuration,
- enable-asadmin-recorder,
- disable-asadmin-recorder,
- asadmin-recorder-enabled

Set-Aws-Config-Source-Configuration

Sets the configuration for the AWS Secrets configuration source.

Synopsis

```
asadmin [asadmin-options] set-aws-config-source-configuration [--help]
[--regionname=regionname]
[--secretname=secretname]
[--target=server]
[--dynamic={false|true}]
[--enabled={true|false}]
```

Description

The `set-aws-config-source-configuration` provides configuration options for the config source which fetches data from AWS Secrets Manager. Make sure that the password aliases `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` are registered first.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--regionname`: The region name of the AWS Secrets service.

`--secretname`: The name of the secret to fetch as a map of properties.

`--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

`--dynamic`: Configure the config source dynamically, which will not require a restart.
`--enabled`: Enables or disables the config source, which fetches data from AWS Secrets Manager

Example

Example 1 Setting the configuration of the AWS secrets

This example configures the AWS Secrets Cloud Config Source.

```
asadmin set-aws-config-source-configuration --dynamic true --enabled true
--secretName payara/test/key --regionName eu-west-2
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-aws-config-source-configuration

Set-Azure-Config-Source-Configuration

Sets the configuration for an Azure Secrets configuration source.

Synopsis

```
asadmin [asadmin-options] set-azure-config-source-configuration [--help]
[--tenant-id=tenant-id]
[--client-id=client-id]
```

```
[--key-vault-name=key-vault-name]
[--private-key-file=private-key-file]
[--thumbprint=thumbprint]
[--target=server]
[--dynamic=false]
[--enabled={true|false}]
```

Description

The `set-azure-config-source-configuration` provides configuration options for the config source which fetches secrets from Azure Key Vault.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target where the configuration source resides. Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

`--key-vault-name`: Name of the Azure Key Vault to fetch secrets from.

`--tenant-id`: Tenant ID of your Azure Active Directory application.

`--client-id`: The Client (Application) ID of your Azure Active Directory application.

`--thumbprint`: The thumbprint of the certificate(certificate hash) that is tied to your Azure Active Directory application.

`--private-key-file`: The path to the Private key file which will be used for certificate-based authentication to request access token. This file will be copied to the config/ directory.

`--dynamic`: Configure the config source dynamically, which will not require a restart.

`--enabled`: Enables or disables the config source, which fetches data from Azure Key Vault.

Example

Example 1 Setting the configuration of the Azure secrets

This example configures the Azure Secrets Config Source.

```
asadmin set-azure-config-source-configuration --dynamic true --enabled true
--keyVaultName demo-secret-key-vault --tenantID 22b3bb26-e046-42df-9c96-
65dbd72c1c81 --clientID 22b3bb26-e046-42df-9c96-65dbd72c1c81 --thumbprint
84E05C1D98BCE3A5421D225B140B36E86A3D5534 --privateKeyPath path/to/privatekey.pem
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-azure-config-source-configuration

Set-Batch-Runtime-Configuration

Configures the batch runtime.

Synopsis

```
asadmin [asadmin-options] set-batch-runtime-configuration [--help]
[--target target]
[--datasourcelookupname datasource-lookup-name]
[--executorservicelookupname executor-service-lookup-name]
[--schemaName schema-name --tablePrefix table-prefix --tableSuffix table-suffix]
```

Description

The `set-batch-runtime-configuration` subcommand configures the batch runtime. The runtime uses a data source and a managed executor service to execute batch jobs. Batch runtime configuration data is stored in the`config` element in `domain.xml`.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which to configure the batch runtime.

Valid values are as follows:
`server`;; Configures the batch runtime for the default server instance `server` and is the default value. cluster-name;; Configures the batch runtime for every server instance in the cluster. instance-name;; Configures the batch runtime for a particular server instance. **`--datasourcelookupname`**: **`-d`**: The JNDI lookup name of the data source to be used to store job information.

+ The default data source is `jdbc/__TimerPool`.

+ NOTE: Do not change the data source after the first batch job has been submitted to the batch runtime for execution. If the data source must be changed, stop and restart the domain and then make the change before any jobs are started or restarted. However, once the data source has been changed, information stored in the previous data source becomes inaccessible. **`--executorservicelookupname`**: **`-x`**: The JNDI lookup name of the managed executor service to be used to provide threads to jobs. The default managed executor service is `concurrent/__defaultManagedExecutorService`.
The managed executor service can be changed after a batch job has been submitted to the batch runtime without impacting execution of the job. **`--schemaName`**: **`-n`**: The database schema that holds the batch tables.

+ Defaults to `APP` **`--tablePrefix`**: The prefix to apply to the batch table name. **`--tableSuffix`**: Sets the suffix to apply to the batch table name.

Examples

Example 1 Configuring the Batch Runtime

The following example configures the batch runtime for the default server instance to use an existing managed executor service named `concurrent/myExecutor`.

```
asadmin> set-batch-runtime-configuration --executorservicelookupname
concurrent/myExecutor
Command set-batch-runtime-configuration executed successfully.
```

Example 2 Configuring datasource and table data

The following command will set the data source and the prefix and suffix for the DAS:

```
asadmin> set-batch-runtime-configuration --target="server" --datasourcelookupname
="jdbc/BatchJobsDS" --tablePrefix="BATCHJ_" --tableSuffix="_JB"
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- list-batch-jobs,
- list-batch-job-executions,
- list-batch-job-steps,
- list-batch-runtime-configuration

Set-Cdieventbus-Notifier-Configuration

Configures the settings of the CDI Event Bus notifier

Synopsis

```
asadmin [asadmin-options] set-eventbus-notifier-configuration [--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[--loopBack={false|true}]
[--hazelcastEnabled={false|true}]
```

Description

This command can be used to configure the behaviour of the CDI EventBus notifier in Payara Server.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are

configuring the CDI Event Bus notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--enabled`**: Defaults to `false`. Enables or disables the notifier. **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart.

+ Otherwise, a restart is required. **`--filter`**: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE) **`--loopBack`**: Defaults to `false`. Whether events should also be observed on the same instance that triggered the instance or not. **`--hazelcastEnabled`**: Defaults to `false`. Whether to enable the Data Grid to send notification events to external instances.

Examples

Example 1 Enable the CDI Event Bus notifier

```
asadmin set-cdieventbus-notifier-configuration --loopBack=true --dynamic=true
--enabled=true --hazelcastEnabled=true
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-cdieventbus-notifier-configuration

Set-Config-Cache

Provides a way to set the value of the cache duration of Eclipse MicroProfile configuration properties.

Synopsis

```
asadmin [asadmin-options] set-config-cache [--help]
[--duration=<duration.in.second>]
[--target=<target[default:server]>]
```

Description

The `set-config-cache` sets the value of the cache duration of Eclipse MicroProfile configuration properties in the targeted instances.

TIP	A value of zero (`0`) disables caching for MicroProfile configuration properties.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--duration`**: Duration in seconds properties are cached. Defaults to 60. **`--target`**: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Disable Caching

This example disables caching by setting the duration to `0`.

```
asadmin set-config-cache --duration=0 --target=myAppCluster
```

Example 2 Sets the caching duration

This example sets the cache TTL (time to live) for configuration properties to `30` seconds.

```
asadmin> set-config-cache --duration=30 --target=myAppCluster
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m

Set-Config-Dir

Provides a way to configure the **Directory** configuration source.

Synopsis

```
asadmin [asadmin-options] set-config-dir [--help]
[--directory=<full.path.to.dir>]
[--target=<target[default:server]>]
```

Description

The `set-config-dir` sets the directory to be used for the directory config source.

- **Leaf directory cannot start with a dot**, rendering `/home/payara/.secret` an invalid path ("." means hidden on a POSIX filesystem).
- If relative, will look beneath server instance location (usually to be found at `domain-dir/`)
- Defaults to `secrets`, targeting `domain-dir/secrets/`

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand.

`--directory`: Full path to the directory containing configuration files.

+ Leaf directory name cannot start with a dot. Defaults to `secrets`.

`--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Setting the Config Directory.

This example sets the directory to be used for the directory config source.

```
asadmin set-config-dir --directory=/home/payara/secrets --target=myAppCluster
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-config-dir

Set-Config-Ordinal

Provides a way to set the ordinal value of a configuration source.

Synopsis

```
asadmin [asadmin-options] set-config-ordinal [--help]
[--target target]
[--ordinalvalue]
[--source={domain|config|server|application|module|cluster}]
```

Description

The `set-config-ordinal` sets the ordinal value for the given config source. Where multiple configuration properties exist, the property from a source with a higher ordinal number is preferred.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--ordinal`: The value of the ordinal to set. This must be a number greater than `1`.

+ A lower number ordinal means lower order of precedence. **`--source`**: The value of

the source to change. Must be one of: * `domain` * `config` * `server` * `application` * `module` * `cluster`. **`--target`**: Specifies the target where the configuration source resides.

Valid values are as follows:

`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster.

Example

Example 1 Setting the ordinal value

This example sets the ordinal value of the application configuration source to `600`.

```
asadmin set-config-ordinal --ordinal=600 --source=application
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-config-ordinal

Set-Config-Property

Provides a way to set the value of a MicroProfile Configuration property in a specific configuration source.

Synopsis

```
asadmin [asadmin-options] set-config-property [--help]
[--propertyName=<property.name>]
[--propertyValue=<property.val>]
[--source={domain|config|server|application|module|cluster}]
[--sourceName=<source.name>]
[--moduleName=<module.name>]
[--target=<target[default:server]>]
```

Description

The `set-config-property` sets the value of a MicroProfile Configuration property in

one of the built-in config sources. The source is specified with `--source` and, where there is ambiguity, the `--sourceName` and `--moduleName` options can be used.

For example, where the source is server, the `--sourceName` can be used to specify the name of the server where the config property is to be stored.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--propertyName`: The name of the configuration property to set. **`--propertyValue`**: The value of the configuration property to set. **`--source`**: The ConfigSource where the property is to be stored. **`--sourceName`**: The name of the ConfigSource when there may be ambiguity, for example a ConfigSource of type application must specify the name of the application. This property is required for sources of type: config, server, application or module **`--moduleName`**: The name of the module when the ConfigSource is of type module. When this is specified, the sourceName parameter must be provided and must have the name of the application where the module is deployed. **`--target`**: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Setting the configuration property value

This example sets the `JMSBrokerURL` property to `my.jms.hostname` within the configuration source named `myApplication`, belonging to the module `myModule`, and targeting the `myAppCluster` cluster.

```
asadmin set-config-property --propertyName=JMSBrokerURL --propertyValue
=my.jms.hostname --source=module --sourceName=myApplication --moduleName=myModule
```

```
--target=myAppCluster
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- delete-config-property
- get-config-property

Set-Datadog-Notifier-Configuration

Configures the settings of the DataDog notifier.

Synopsis

```
asadmin [asadmin-options] set-datadog-notifier-configuration [--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[--key key]
```

Description

This command can be used to configure the behaviour of the DataDog notifier in Payara Server.

IMPORTANT	This command is available as long as the Datadog notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--target`: This option helps specify the target on which you are configuring the DataDog notifier. Valid values are:

`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--enabled`**: Defaults to `false`. Enables or disables the log notifier **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart.

+ Otherwise, a restart is required. **`--filter`**: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE) **`--key`**: Defines the license key to be used by the notifier.

Examples

Example 1 Enable the DataDog notifier and set its license key

```
asadmin set-datadog-notifier-configuration --dynamic=true --key
=af510f122026bc9107f7c5f4f049e597 --enabled=true
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-datadog-notifier-configuration

Set-Discord-Notifier-Configuration

Configures the settings of the Discord notifier.

Synopsis

```
asadmin [asadmin-options] set-discord-notifier-configuration [--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[--webhookId id]
[--webhookToken token]
```

Description

This command can be used to configure the behaviour of the Discord notifier in Payara

Server.

IMPORTANT	This command is available as long as the Discord notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the Discord notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--enabled`**: Defaults to `false`. Enables or disables the log notifier **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart.

+ Otherwise, a restart is required. **`--filter`**: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE) **`--webhookId`**: The ID of the webhook required to authenticate to the corresponding Discord channel. **`--webhookToken`**: The token required to authenticate the webhook to the corresponding Discord channel.

Examples

Example 1 Enable the Discord notifier and set its required webhook settings

```
asadmin set-discord-notifier-configuration --enabled=true --dynamic=true
--webhookId=xxx --webhookToken=yyy
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-discord-notifier-configuration

Set-Dynamodb-Config-Source-Configuration

Sets the configuration for the DynamoDB Config Source.

Synopsis

```
asadmin [asadmin-options] set-dynamodb-config-source-configuration [--help]
[--region-name=region-name]
[--table-name=table-name]
[--key-column-name=key-column-name]
[--value-column-name=value-column-name]
[--limit=100]
[--target=server]
[--dynamic={false|true}]
[--enabled={true|false}]
```

Description

The `set-dynamodb-config-source-configuration` provides configuration options to set the properties required by DynamoDB config source to retrieves data from the configured DynamoDB database table.

> **TIP** Make sure that the password aliases `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` are created first.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

`--region-name`: The regional location of your DynamoDB service.

`--table-name`: Name of the table in the database.

`--key-column-name`: Name of the column containing the key.

`--value-column-name`: Name of the column containing the value.

`--limit`: The maximum number of items to retrieves from the database table. The default value is 100.

`--dynamic`: Configure the config source dynamically, which will not require a restart.

`--enabled`: Enables or disables the config source.

Example

Example 1 Setting the configuration of the DynamoDB Config Source

This example configures the DynamoDB Config Source.

```
asadmin> set-dynamodb-config-source-configuration --enabled true --dynamic true
--region-name eu-west-2 --table-name Books --key-column-name ISBN --value-column
-name Title --limit 10
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-dynamodb-config-source-configuration

Set-Ejb-Invoker-Configuration

Enables or disables the endpoint that allows HTTP(S) transport based communication for remote EJBs.

Synopsis

```
asadmin [asadmin-options] set-ejb-invoker-configuration [--help]
[--target] target
[--enabled={true|false}]
[--securityenabled={true|false}]
```

```
[--roles] roles-list
[--authmoduleclass] sam-class
[--authmodule] security-provider-id
[--realmname] realm-name
[--authtype] auth-type
[--endpoint ] context-root
[--virtualservers] virtualserver-list
```

Description

The `set-ejb-invoker-configuration` can be used to configure the Remote EJB Lite HTTP endpoint settings.

IMPORTANT	The `set-ejb-invoker-configuration` `--enabled=true/false` commands actually deploy/undeploy an internal small WAR application that exposes the endpoint. This application is shown in all overviews that show deployed applications that are currently enabled.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which to configure the `ejb-invoker` endpoint. Valid values are as follows:
`server`;; Configures the EJB invoker endpoint for the default server instance `server` and is the default value. cluster-name;; Configures the EJB invoker endpoint for every server instance in the cluster. deployment-group-name;; Configures the EJB invoker endpoint for every server instance in the deployment group. instance-name;; Configures the EJB invoker endpoint for a particular server instance. **`--enabled`**: Enables or disables the `ejb-invoker` application.

+ Defaults to `true` **`--securityenabled`**: Whether secure access to the `ejb-invoker` endpoint is enabled or not.

+ Defaults to `false` **`--roles`**: If defined, the endpoint will be configured to be authorised for users that belong to the list of the roles specified (in comma-separated format). **`--endpoint`**: The context root used to expose the `ejb-invoker` application

endpoint.

+ Defaults to `ejb-invoker` **`--authmoduleclass`**: Defines the full qualified class name implementing a `jakarta.security.auth.message.module.ServerAuthModule` subclass.

+ If defined and `securityenabled` is set to `true`, the HttpServlet Message security provider is created and `ejb-invoker` application secured with the configured authentication module. **`--authmodule`**: Defines the existing message security provider ID.

+ If defined and `securityenabled` is set to `true`, the `ejb-invoker application` will be secured with it. **--realmname`**: Defines the registered realm name.

+ If defined and `securityenabled` it set to `true`, the `ejb-invoker` application will be secured with the corresponding realm that will be used for authentication. -- **`authtype`**: If defined, the endpoint will be assigned to a list of **virtual servers** specified as a comma-separated list of names.

+ Otherwise, the endpoint will be assigned to all virtual servers available.

Examples

The following command will enable the endpoint on the DAS:

```
asadmin> set-ejb-invoker-configuration --enabled=true
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-ejb-invoker-configuration

Set-Email-Notifier-Configuration

Configures the settings of the Email notifier.

Synopsis

```
asadmin [asadmin-options] set-email-notifier-configuration [--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[--jndiName sessionName]
[--recipient recipient]
```

Description

This command can be used to configure the behaviour of the Email notifier in Payara Server.

IMPORTANT	This command is available as long as the Email notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the Email notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--enabled`**: Defaults to `false`. Enables or disables the log notifier **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart.

+ Otherwise, a restart is required. **`--filter`**: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE) **`--jndiName`**: Defines the JNDI name of the mail session resource used to send email notifications. **`--recipient`**: The recipient's email address that will receive the notification events.

Examples

Example 1 Enable the Email notifier and configure it using a mail session

```
asadmin set-email-notifier-configuration --jndiName=mail/sampleSession --recipient
=notifications@example.com --enabled=true --dynamic=true
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-email-notifier-configuration

Set-Environment-Warning-Configuration

Allows customization of a warning bar for clear identification of the server environment in the Admin Console.

Synopsis

```
asadmin [asadmin-options] set-environment-warning-configuration [--help]
[--enabled={true|false}]
[--message=message]
[--backgroundcolour=backgroundcolour]
[--textcolour=textcolour]
```

Description

The `set-environment-warning-configuration` subcommand sets the environment warning properties and can enable/disable the warning.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. --**enabled**: Activates or deactivates the Environment Warning feature, enabling or disabling the display of the warning bar in the Admin Console based on the chosen setting. --**message**: Specifies the custom text message to be displayed within the Environment Warning bar, allowing for tailored alerts or reminders specific to the current server environment. --**backgroundcolour**: Sets the background color of the

Environment Warning bar, providing a visual means to distinguish between different environments by color-coding the warning bar. **--textcolour**: Determines the color of the text within the Environment Warning bar, ensuring readability and contrast against the background color, enhancing the visibility of the warning message.

Examples

Example 1

```
asadmin> set-environment-warning-configuration --enabled=true --message="Caution,
this is on production" --textcolour="#00ff00" --backgroundcolour="#ff00ff"
Command set-environment-warning-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-environment-warning-configuration

Set-Eventbus-Notifier-Configuration

Configures the settings of the Event Bus notifier

Synopsis

```
asadmin [asadmin-options] set-eventbus-notifier-configuration [--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[--topicName topicName]
```

Description

This command can be used to configure the behaviour of the EventBus notifier in Payara Server.

NOTE	The Data Grid must be enabled for the Event Bus to be available.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the Event Bus notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--enabled`**: Defaults to `false`. Enables or disables the notifier. **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart.

+ Otherwise, a restart is required. **`--filter`**: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE) **`--topicName`**: Defines the name of the internal Hazelcast distributed topic which will be used by the event bus to subscribe to notification events.

Examples

Example 1 Enable the Event Bus notifier and set its topic

```
asadmin set-eventbus-notifier-configuration --topicName=testTopic --dynamic=true
--enabled=true
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-eventbus-notifier-configuration

Set-Fault-Tolerance-Configuration

Modifies the configuration settings of the MicroProfile Fault Tolerance service.

Synopsis

```
asadmin [asadmin-options] set-fault-tolerance-configuration [--help]
```

```
[--managedexecutorservicename=managedexecutorservicename]
[--managedscheduledexecutorservicename=managedscheduledexecutorservicename]
[--target=server-config]
```

Description

The `set-fault-tolerance-configuration` provides a way to modify the configuration of the MicroProfile Fault Tolerance service in the specified target.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--managedexecutorservicename`: The Logical JNDI name of the Managed Executor Service to look up. Changes to this configuration require a restart of the server to take effect.

+ Defaults to `java:comp/DefaultManagedExecutorService`

`--managedscheduledexecutorservicename`: The Logical JNDI name of the Managed Scheduled Executor Service to look up. Changes to this configuration require a restart of the server to take effect.

+ Defaults to `java:comp/DefaultManagedScheduledExecutorService`

`--target`: Specifies the target of this command.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

Example

Example 1 Setting MicroProfile Fault Tolerance configuration

This example sets the configuration of the MicroProfile Fault Tolerance service.

```
asadmin> set-fault-tolerance-configuration --managedexecutorservicename
```

```
=java:comp/DefaultManagedExecutorService --target instance1
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-fault-tolerance-configuration

Set-Gcp-Config-Source-Configuration

Sets the configuration for the GCP secrets configuration source.

Synopsis

```
asadmin [asadmin-options] set-gcp-config-source-configuration [--help]
[--projectname=projectname]
[--jsonkeyfile=jsonkeyfile]
[--target=server]
[--dynamic={false|true}]
[--enabled={true|false}]
```

Description

The `set-gcp-config-source-configuration` provides configuration options for the config source which fetches data from GCP Secrets Manager.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--projectname`: The name of the GCP project to fetch secrets from.

`--jsonkeyfile`: The path to the JSON key file which a service account uses to authenticate with GCP Secrets. This file will be copied to the config/ directory.

`--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server

instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

`--dynamic`: Configure the config source dynamically, which will not require a restart.

`--enabled`: Enables or disables the config source.

Example

Example 1 Setting the configuration of the GCP Config Source

This example configures the GCP Config Source.

```
asadmin set-gcp-config-source-configuration --dynamic true --enabled true
--jsonKeyFile /path/to/token/file.json --projectName project-name
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-gcp-config-source-configuration

Set-Hashicorp-Config-Source-Configuration

Sets the configuration for the HashiCorp secrets Config Source.

Synopsis

```
asadmin [asadmin-options] set-hashicorp-config-source-configuration [--help]
[--api-version=2]
[--secrets-engine-path=secrets-engine-path]
[--secrets-path=secrets-path]
[--vault-address=vault-address]
[--target=server]
[--dynamic={false|true}]
[--enabled={true|false}]
```

Description

The `set-hashicorp-config-source-configuration` provides configuration options for the Hashicorp configuration source which fetches secrets from HashiCorp

Vault. Hashicorp config source only supports the Vault KV secrets engine.

Make sure that the password aliases `HASHICORP_VAULT_TOKEN` is registered.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--api-version`: KV Secrets Engine API version.

`--secrets-engine-path`: The path where the KV secrets engine is enabled.

`--secrets-path`: The path where your secrets are stored.

`--vault-address`: Vault's address represented as URL and port. For example, "http://127.0.0.1:8200".

`--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

`--dynamic`: Configure the config source dynamically, which will not require a restart.

`--enabled`: Enables or disables the config source.

Example

Example 1 Setting the configuration of the HashiCorp Config Source

This example configures the HashiCorp Config Source.

```
asadmin set-hashicorp-config-source-configuration --enabled true --dynamic true
--api-version 2 --vault-address https://127.0.0.1:8200 --secrets-engine-path secret
--secrets-path my-secret
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-hashicorp-config-source-configuration

Set-Hazelcast-Configuration

Configures the underlying Data Grid (Hazelcast) instance.

Synopsis

```
asadmin [asadmin-options] set-hazelcast-configuration
[--help]
[--target target]
[--enabled={true|false}]
[--dynamic={true|false}]
[--hazelcastconfigurationfile hazelcastconfigurationfile]
[--startport startport]
[--publicaddress publicaddress]
[--daspublicaddress daspublicaddress]
[--dasbindaddress dasbindaddress]
[--dasport dasport]
[--clustermode clustermode]
[--tcpipmembers tcpipmembers]
[--dnsmembers dnsmembers]
[--interfaces interfaces]
[--multicastgroup multicastgroup]
[--multicastport multicastport]
[--clustername clustername]
[--jndiname name]
[--cachemanagerjndiname name]
[--cachingproviderjndiname name]
[--executorpoolsize executorpoolsize]
[--executorqueuecapacity executorqueuecapacity]
[--scheduledexecutorpoolsize scheduledExecutorPoolsize]
[--scheduledexecutorqueuecapacity scheduledexecutorQueueCapacity]
[--licensekey=licensekey]
[--lite={true|false}]
[--hostawarePartitioning={true|false}]
[--membername membername]
[--membergroup membergroup]
[--kubernetesNamespace namespace]
[--kubernetesServicename serviceName]
[--autoIncrementport={true|false}]
[--configSpecificDatagridStartport startPort]
[--encryptDatagrid={true|false}]
```

Description

This command can be used to configure the settings of the underlying Data Grid (Hazelcast) instance.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are executing this operation. Valid values are:
`server`;; Applies to the default server instance. This is the default value.
instance_name;; Applies to a specified server instance. **`--enabled`**: Enables the Data Grid (Hazelcast).

+ If no `target` is specified, this will enable Hazelcast on the domain configuration `server-config`.

+ If no arguments are specified, defaults to `true`. If other options are specified, asadmin will interactively prompt for `--enabled`.

+ The default value is `true`. **`--dynamic`**: Enable or disable dynamic stopping and starting of the embedded Hazelcast member. The default value is `false`. **`--hazelcastconfigurationfile`**: **`-f`**: The Hazelcast configuration file to use.

+ This path is relative to the domain config directory, `domain-dir/config/`.

+ The default value is `hazelcast-config.xml`. **`--startport`**: The port to run the Data Grid (Hazelcast) on for Payara Server Instances.

+ If this port is busy, the port specified will be incremented until a valid port is found.

+ The default value is `5900`. **`--publicaddress`**: The Public Address to use for this Data Grid Node.

+ Used when the node is behind a NAT. **`--daspublicaddress`**: This is the public IP Address that the DAS should use if the DAS is behind a NAT firewall and other Payara

Server Instances need to connect via the public NAT address rather than the internal IP Address. `--dasbindaddress`: Explicitly specifies the IP Address the DAS should bind the Hazelcast instance to.

+ By default, Hazelcast will try to choose the correct bind address from the IP addresses available on the host.

+ If you find the DAS is binding to an incorrect IP Address the correct address can be specified here. `--dasport`: The port to run Hazelcast on for the DAS.

+ If this port is busy, the port specified will be incremented until a valid port is found.

+ The default value is `4900`. `--clustermode`: Sets the Data Grid cluster discovery mode.

+ Can be one of `domain`, `tcpip`, `multicast` or `kubernetes`.

+ See domain-discovery-mode for a detailed description of each mode.
`--tcpipmembers`: Comma separated list of `IP-Address:port` used to discover cluster members if `tcpip` is selected as the cluster mode. `--dnsmembers`: Sets the list of DNS names and ports of all Data Grid members when dns is used.
`--interfaces`: Similar to DAS Bind address by default Hazelcast will choose the correct IP Address to bind to.

+ If you find Hazelcast is binding to the incorrect address a comma separated list of IP Addresses can be specified here. `--multicastgroup`: `-g`: The multicast group for communications in the Hazelcast instance if `multicast` is selected as cluster mode. The default value is `224.2.2.3`. `--multicastport`: The multicast port for communications in the Hazelcast instance if `multicast` is selected as cluster mode. The default value is `54327`. `--clustername`: Specifies the Hazelcast group name, to be optionally used to help divide clusters into logical, segregated groups (i.e. `dev-group`, `prod_group`). `--jndiname`: `-j`: Specifies the JNDI name to bind the Hazelcast instance to. `--cacheManagerJndiname`: Specified the JNDI name to bind the JCache Caching provider. `--cachingProviderJndiname`: The JNDI name to bind

the JCache Caching Provider to. `--executorpoolsize`: The thread pool's size for the Hazelcast Executor service. `--executorqueuecapacity`: Queue Capacity of the Data Grid Executor Service.

+ Executions are queued until a thread is available. `--scheduledexecutorpoolsize`: The thread pool's size for the Hazelcast Scheduled Executor service. `--scheduledexecutorqueuecapacity`: Queue Capacity of the Data Grid Scheduled Executor Service. Executions are queued until a thread is available. `--licensekey`: `-lk`: Used to set the license key to start Hazelcast using an equivalent Hazelcast Enterprise edition compatible build.

+ IMPORTANT: Hazelcast Enterprise binaries have to be properly patched in the server installation for the Data Grid to be started correctly. `--lite`: Sets the cluster to lite mode, where it will cluster but **not** store any cluster data. The default value is `false`. `--hostawarePartitioning`: Whether to enable host-aware partitioning for the cluster.

+ Host aware partitioning **must be** enabled on all members of the cluster (including Payara Micro instances) for this feature to work correctly. The default value is `false`. `--membername`: Member name. The default value is `instance name`. `--membergroup`: Instance group name for all Server Instances using the configuration. Future functionality may use this name. The default value is `config name`. `--kubernetesNamespace`: Sets the Kubernetes namespace when kubernetes is used.

+ NOTE: If a namespace is not specified, Payara Server will use the `default` namespace.

`--kubernetesServicename`: Sets the Kubernetes service name when kubernetes is used. `--autoincrementport`: By default, the Data Grid uses the next unoccupied port that is available starting with the start port.

+ When auto-increment is turned off an occupied start port results in a startup failure instead.

+ The default value is `false`. **`--configSpecificDataGridStartPort`**: Sets a specific Data Grid start port to the instance.

+ NOTE: This argument is only applicable when targeting standalone instances.

+ If set to `0` the Domain wide start port will be used instead. **`--encryptdatagrid`**: Enables or disables encryption of data stored in the "Domain Data Grid" store using a key created using the "`generate-encryption-key`" command.

+ The default value is `false`.

Examples

Example 1 Basic configuration of the Data Grid settings

This example demonstrates configuring the Data Grid settings for a Hazelcast cluster called `cluster1`.

```
asadmin> set-hazelcast-configuration --enabled=true --target=cluster1 --dynamic
=true -f hazelcast-config.xml --startPort=5902 -g 224.2.2.3 --multicastPort=6666
--clustermode multicast --hostawarePartitioning=true -j payara/Hazelcast
Command set-hazelcast-configuration executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Domain Data Grid",
- "`list-hazelcast-cluster-members`",
- "`list-hazelcast-members`",
- "`restart-hazelcast`",
- "`list-cache-keys`",
- "`list-caches`",

- "`clear-cache`",
- "`get-hazelcast-configuration`",
- "`generate-encryption-key`"

Set-Healthcheck-Configuration

Configures the Healthcheck service.

Synopsis

```
asadmin [asadmin-options] set-healthcheck-configuration
[--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--historic-trace-enabled={false|true}]
[--historic-trace-store-size size]
[--historic-trace-store-timeout integer.value {s|m|h|d}]
[--set-notifiers notifiers]
[--enable-notifiers notifiers]
[--disable-notifiers notifiers]
```

Description

Enables and disables the HealthCheck service. This includes configuration for tracing historic health check events for later inspection.

This command can also be used to configure which notifiers will be used to broadcast notification events from the Healthcheck service.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the Healthcheck service. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--enabled`**: Defaults to `false`. Enables or disables the healthcheck service **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart. **`--historic-trace-enabled`**: Defaults to `false`. Enables storing

traces in a rolling store for later inspection. `--historic-trace-store-size`: Defaults to `20`. Sets the maximum number of health checks to store. `--historic-trace-store-timeout`: Sets the time period after which a historic health check event entry is removed from history.

+ The time expression should consist of a number followed by a time unit; `s` for seconds, `m` for minutes, `h` for hours or `d` for days.

+ If no time unit is given the number specifies seconds.

+ If the parameter is zero or unspecified there is no timeout for entries. `--set-notifiers`: Use the option to set the notifiers that will receive notifications generated by the Healthcheck Service.

+ A comma-separated list can be used to represent multiple notifiers.

+ TIP: This option will replace all the notifiers that have already been set to the HealthCheck Service. `--enable-notifiers`: Use the option to enable one or more notifiers for the Healthcheck service.

+ A comma-separated list can be used to represent multiple notifiers. `--disable-notifiers`: Use the option to disable one or more notifiers for the Healthcheck service.

+ A comma-separated list can be used to represent multiple notifiers.

Examples

Example 1 Basic Configuration

The following example will enable the Healthcheck service such that it will only activate from the next time the server is restarted.

It sets the log notifier and JMS notifiers and the historical trace store to retain 20 health checks.

```
asadmin set-healthcheck-configuration --enabled=true --dynamic=false --historic
```

```
-trace-enabled=true --historic-trace-store-size=20 --set-notifiers=log-
notifier,jms-notifier
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "`set-healthcheck-service-configuration`",
- "`get-healthcheck-configuration`",
- "`list-healthcheck-services`"

Set-Healthcheck-Service-Configuration

Enables or disables the Healthcheck of a specific performance metric service.

Synopsis

```
asadmin [asadmin-options] set-healthcheck-service-configuration
[--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--service=serviceName]
[--checker-name=checkerName]
[--add-to-microprofile-health={true|false}]
[--time=time]
[--time-unit={DAYS|HOURS|MINUTES|SECONDS|MILLISECONDS}]
[--threshold-critical=threshold]
[--threshold-warning=threshold]
[--threshold-good=threshold]
[--hogging-threads-threshold=threshold]
[--hogging-threads-retry-count=count]
[--stuck-threads-threshold=threshold]
[--stuck-threads-threshold-unit={DAYS|HOURS|MINUTES|SECONDS|MILLISECONDS}]
[--add-metric=metricName]
[--delete-metric=metricName]
```

Description

Enables and disables the monitoring of a HealthCheck performance metric service. The command also configures the frequency of monitoring for that metric.

The command can also be used to configures performance metric specific properties.

WARNING	If this command is run before running the `set-healthcheck-configuration` command, it will succeed and the configuration will be saved, but the Healthcheck service will not be enabled until the command is used.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the Healthcheck performance metric service. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--enabled`**: Defaults to `false`. Enables or disables the healthcheck performance metric service **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart. **`--service`**: The performance service metric name. Required.

+ One of:

- `connection-pool` or `cp`
- `cpu-usage` or `cu`
- `garbage-collector` or `gc`
- `heap-memory-usage` or `hmu`
- `hogging-threads` or `ht`
- `machine-memory-usage` or `mmu`
- `stuck-thread` or `st`
- `mp-health` or `mh`
- `mp-metrics` or `mm`

`--checker-name`: A user determined name for easy identification of the checker.

+ This should be unique among the services you have configured, to avoid confusion on the notification messages.

+ Depends on the service checker. One of:

- `CONP`
- `CPUC`
- `GBGC`
- `HEAP`
- `HOGT`
- `MEMM`
- `MP`
- `MPM`

`--add-to-microprofile-health`: Defaults to `false`. When enabled, the checker results are displayed on MicroProfile Health rest-endpoints as a readiness check.

`--time`: Defaults to `5`. The amount of time units that the service will use to periodically monitor the metric. **`--time-unit`**: Defaults to `MINUTES`. The time unit to set the frequency of the metric monitoring.

+ NOTE: Must correspond to a valid `java.util.concurrent.TimeUnit`
`--threshold-critical`: Defaults to `90`. The threshold value that this metric must surpass to generate a **`CRITICAL`** event.

+ A value between *WARNING VALUE* and `100` must be used.

+ NOTE: Available for the `cp`, `cu`, `gc`, `hmu` and `mmu` metric services. **`--threshold-warning`**: Defaults to `50`. The threshold value that this metric must surpass to generate a **`WARNING`** event.

+ A value between *GOOD VALUE* and *WARNING VALUE* must be used.

+ NOTE: Available for the `cp`, `cu`, `gc`, `hmu` and `mmu` metric services. **`--threshold-good`**: Defaults to `0`. The threshold value that this metric must surpass to generate a **`GOOD`** event.

+ A value between `0` and *WARNING VALUE* must be used.

+ NOTE: Available for the `cp`, `cu`, `gc`, `hmu` and `mmu` metric services.

`--hogging-threads-threshold`: Defaults to `95`. The threshold value that this metric will be compared to mark threads as hogging the CPU.

+ NOTE: Only available for the `ht` metric service. **`--hogging-threads-retry-count`**: Defaults to `3`. The number of retries that the checker service will execute in order to identify a hogging thread.

+ NOTE: Only available for the `ht` metric service. **`--stuck-threads-threshold`**: The threshold above which a thread is considered stuck. Must be `1` or greater.

+ NOTE: Only available for the `st` metric service. **`--stuck-threads-threshold-unit`**: The unit for the threshold for when a thread should be considered stuck.

+ NOTE: Must correspond to a valid `java.util.concurrent.TimeUnit`

+ NOTE: Only available for the `st` metric service.

`--add-metric`: Adds a metric exposed by Eclipse MicroProfile Metrics to monitor.

+ Takes a string of the format `'metricName=MetricName description=Description'`, where `metricName` is a mandatory value.

`--delete-metric`: Removes a metric exposed by MicroProfile Metrics that has been added to monitor. Takes a string of the format `'metricName=MetricName'`, where `metricName` is required.

Examples

Example 1 Basic Configuration

A very basic example command to simply enable the Garbage Collector checker service and activate it without needing a restart would be as follows:

```
asadmin set-healthcheck-service-configuration --enabled=true --service=gc --
dynamic=true
```

Example 2 Setting up thresholds for configuring JDBC Connection Pools health monitoring

Monitoring the health of JDBC connection pools is a common need. In that scenario, it is very unlikely that on-the-fly configuration changes would be made, so a very high `CRITICAL` threshold can be set.

Likewise, a nonzero `GOOD` threshold is needed because an empty or unused connection pool may not be healthy either.

The following command would apply these settings to the connection pool checker:

```
asadmin set-healthcheck-service-configuration --service=cp --dynamic=true
--threshold-critical=95 --threshold-warning=70 --threshold-good=30
```

Example 3 Setting up thresholds for identifying CPU hogging threads

Monitoring which threads hog the CPU is extremely important since this can lead to performance degradation, deadlocks and extreme bottlenecks issues that web applications can incur.

In some cases the defaults are all that is needed, but imagine that in a critical system you want to set the threshold percentage to **90%**, and you want to make sure that the health check service guarantees the state of such threads with a retry count of **5**.

Additionally, you want to set the frequency of this check for every *20 seconds*.

The following command would apply these settings to the Hogging threads checker service:

```
asadmin> set-healthcheck-service-configuration --service=ht --dynamic=true
```

```
--hogging-threads-threshold=90 --hogging-threads-retry-count=5 --time=20 --time
-unit=SECONDS
```

Example 4 Setting up thresholds for identifying stuck threads

The following example configures the stuck threads checker to check every 30 seconds for any threads which have been stuck for more than 5 minutes and applies the configuration change without needing a restart:

```
asadmin set-healthcheck-service-configuration --service=st --enabled=true --
dynamic=true --time=30 --time-unit=SECONDS --stuck-threads-threshold=5 --stuck
-threads-threshold-unit=MINUTES
```

Example 5 Setting up thresholds adding a MicroProfile Metrics check for the maximum thread count:

The following example configures the Microprofile Metrics Checker to add `base_thread_max_count` metrics for monitoring, adds the checker to MicroProfile Health to display its result on MicroProfile Health rest-endpoints and applies the configuration change without needing a restart:

```
asadmin set-healthcheck-service-configuration --service=mp-metrics --enabled=true
--dynamic=true --add-to-microprofile-health=true --add-metric
='metricName=base_thread_max_count'
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "`set-healthcheck-configuration`",
- "`get-healthcheck-configuration`"

Set-Jdbc-Config-Source-Configuration

Configures the JDBC configuration source.

Synopsis

```
asadmin [asadmin-options] set-jdbc-config-source-configuration [--help]
[--jndiname=jndiname]
[--tablename=tablename]
[--keycolumnname=keycolumnname]
[--valuecolumnname=valuecolumnname]
[--target=server]
```

Description

The `set-jdbc-config-source-configuration` provides configuration options to set the properties required by JDBC config source to read the values from the configured database.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--jndiName`: JNDI (Java Naming and Directory Interface) name under which the JDBC resource will be registered in the application server's JNDI namespace. `--tablename`: Name of the table in the database. `--keycolumnname`: Name of the column in the specified table that serves as the key for retrieving data. `--valuecolumnname`: Name of the column in the specified table that contains the values associated with the keys. `--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster.

Example

Example 1 Configuring the JDBC Config Source.

This example configures the JDBC config source.

```
asadmin set-jdbc-config-source-configuration --jndiName jndi-name --tableName Books
--keyColumnName ISBN --valueColumnName Title
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-jdbc-config-source-configuration

Set-Jms-Notifier-Configuration

Configures the settings of the JMS notifier

Synopsis

```
asadmin [asadmin-options] set-jms-notifier-configuration [--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[--contextFactoryClass className]
[--connectionFactoryName connectionFactoryName]
[--queueName queueName]
[--url url]
[--userName userName]
[--password password]
```

Description

This command can be used to configure the behaviour of the JMS notifier in Payara Server Full Profile.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the JMS notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--enabled`**: Defaults to `false`. Enables or disables the notifier **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart.

+ Otherwise, a restart is required. `--filter`: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE) `--contextFactoryClass`: Defines the naming context factory class used to provision the JMS resources required to send the notification events.

+ NOTE: The referenced class name should point to a valid implementation of `javax.naming.spi.InitialContextFactory` and should be located in the server's classpath.

+ TIP: In most cases, using the default context factory class (`com.sun.enterprise.naming.SerialInitContextFactory`) is enough. `--connectionFactoryName`: Defines the JNDI name of an existing JMS connection factory which will be used to send the notification messages. `--queueName`: Defines the physical name of an existing JMS queue destination used to send the messages to. `--url`: Defines the URL of a JMS broker server that will handle the sending and reception of the notification message. `--userName`: Defines the user that will authentication to the JMS broker server (if required). `--password`: Defines the password credential of the user that will authentication to the JMS broker server (if required).

Examples

Example 1 Enable the JMS Notifier and Configure it using OpenMQ

This example enables the JMS notifier and sets it up to connect to the default, local OpenMQ broker server and use a queue named `testQueue`:

```
asadmin set-jms-notifier-configuration --dynamic=true --enabled=true
--contextFactoryClass=com.sun.enterprise.naming.SerialInitContextFactory
--connectionFactoryName=jms/__defaultConnectionFactory --queueName=testQueue --url
=localhost:7676 --username=testUser --password=******** --target=server-config
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-jms-notifier-configuration

Set-Jmx-Monitoring-Configuration

Configures the JMX monitoring service subsystem.

Synopsis

```
asadmin [asadmin-options] set-jmx-monitoring-configuration
[--help]
[--target target]
[--enabled={false|true}]
[--logFrequency frequency]
[--logFrequencyUnit={DAYS|HOURS|MINUTES|SECONDS|MILLISECONDS}]
[--addAttribute="attributes"]
[--delAttribute="name"]
[--set-notifiers notifiers]
[--enable-notifiers notifiers]
[--disable-notifiers notifiers]
```

Description

This command can be used to configure the settings of the JMX Monitoring subsystem.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the JMX monitoring service. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--enabled`**: Defaults to `false`. Enables or disables the JMX monitoring service **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart. **`--addAttribute`**: Provides a way to add a new **MBean** attribute to monitor using the service.

+ This argument takes in a string of space-delimited key-value pairs corresponding to the values of a JMX MBean definition. The `attributeName` and `objectName` fields are required, but `description` is not.

+ For example, providing `attributeName=HeapMemoryUsage` denotes that the name of the **MBean** attribute to log is `HeapMemoryUsage`, while `objectName=java.lang:type=Memory` denotes the `ObjectName` of the **MBean** to look for the attribute on is `java.lang:type=Memory`. **`--delAttribute`**: Provides a way to remove an **MBean** attribute that is being monitored by the JMX Monitoring service.

+ Use the name of the attribute that you want to remove from the monitoring output. **`--logFrequency`**: Defaults to `15`. The frequency that the service will use to periodically log monitoring entries. **`--logFrequencyUnit`**: Defaults to `SECONDS`. The time unit to set the frequency of the log done by the JMX Monitoring service. **`--set -notifiers`**: Use the option to set the notifiers that will receive notifications generated by the JMX Monitoring service.

+ A comma-separated list can be used to represent multiple notifiers.

+ TIP: This option will replace all the notifiers that have already been set to the JMX Monitoring Service. **`--enable-notifiers`**: Use the option to enable one or more notifiers for the JMX Monitoring service.

+ A comma-separated list can be used to represent multiple notifiers. **`--disable -notifiers`**: Use the option to disable one or more notifiers for the JMX Monitoring service.

+ A comma-separated list can be used to represent multiple notifiers.

Examples

Example 1 Enables JMX monitoring service along with the AMX subsystem

```
asadmin set-jmx-monitoring-configuration --enabled=true --dynamic=true --target
=server
```

Example 2 Enables JMX monitoring service and monitors the amount of used Heap memory of the server's JVM

```
asadmin set-jmx-monitoring-configuration --addAttribute
'attributeName=HeapMemoryUsage.used objectName=java.lang:type=Memory' --enabled
true
```

Example 3 Enables JMX monitoring service and configures the logging frequency to 5 minutes

```
asadmin set-jmx-monitoring-configuration --logFrequency 5 --logFrequencyUnit
MINUTES --enabled true
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Monitoring Basics"
- "`set-amx-enabled`"
- "`get-jmx-monitoring-configuration`"

Set-Ldap-Config-Source-Configuration

Sets the LDAP Config Source

Synopsis

```
asadmin [asadmin-options] set-ldap-config-source-configuration [--help]
[--enabled={true|false}]
[--dynamic={true|false}]
[--url=<ldap-server-url>]
[--authType={simple|none}]
[--startTLSEnabled={true|false}]
[--bindDN=<user-distinguished-name>]
[--bindDNPassword=<user-credential>]
[--searchBase={subtree|onelevel|object}]
[--searchFilter=<search-filter>]
[--searchScope=<search-scope>]
[--connectionTimeout=<integer-value-in-milliseconds>]
[--readTimeout=<integer-value-in-milliseconds>]
[--target=<target[default:server]>]
```

Description

The `set-ldap-config-source-configuration` provides a way to configure the

MicroProfile LDAP configuration source.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--enabled`**: Defaults to `false`. Enables or disables the config source **`--dynamic`**: Defaults to `false`. If set to true, applies the changes instantly without a restart. Otherwise, a restart is required.

`--url`: URL where the LDAP server can be reached

`--authType`: Defaults to `none`. Type of the Authentication method used during the connection with the LDAP server.

+ The MicroProfile LDAP config source supports simple and none auth type. If simple auth type is selected then Bind DN and Bind DN Password are mandatory. If none auth type is selected then **Search Base** and **Search Filter** are mandatory otherwise optional.

+ If **Search Base** and **Search Filter** are not defined then data fetched from direct binding.

`--startTLSEnabled`: Defaults to `false`. Secure the LDAP server connection using LDAP over TLS (STARTTLS) protocol

`--bindDN`: Distinguished name for the application that will be used to access the LDAP server

`--bindDNPassword`: Password for the application defined by the Bind DN member. In place of the password, you can enter an alias which corresponds to the password in the form: ${ALIAS=<password-alias-name>}

`--searchBase`: Overrides 'Base DN' and causing the search to be used instead of direct binding

`--searchFilter`: Search filter to find values starting from the 'Search Base DN' with the scope specified by 'Search Scope'.

`--searchScope`: Search scope determines the depth of the search in the LDAP tree

`--connectionTimeout`: The timeout for connecting to the LDAP server in milliseconds. The value should be greater than zero. If this value is not specified or less than one, then wait for the response until it is received.

`--readTimeout`: The timeout for fetching the results from the LDAP server in milliseconds. The value should be greater than zero. If this value is not specified or less than one, then wait for the response until it is received.

`--target`: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster.

Example

Example 1 Setting the LDAP configuration

This example enables and configures LDAP authentication on the Payara Server instance, specifying the LDAP server's URL, authentication type, binding credentials and other settings.

```
asadmin set-ldap-config-source-configuration --enabled=true --dynamic=true --url
=ldap://ldap.forumsys.com:389 --authType=simple --bindDnPassword=password --bindDn
=cn=read-only-admin,dc=example,dc=com
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-ldap-config-source-configuration

Set-Log-Attributes

Sets the logging attributes for one or more loggers.

Synopsis

```
asadmin [asadmin-options] set-log-attributes [--help]
[--target=target]
attribute-name=attribute-value[:attribute-name=attribute-value]*
```

Description

The `set-log-attributes` subcommand sets logging attributes for one or more loggers. The attributes you can set correspond to the attributes that are available in the `logging.properties` file for the domain.Depending on the attributes you set, a server restart may be necessary.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: The server domain, instance, or cluster for which logger attributes will be set. If this option is omitted, attributes are set for the default server.

Operands

target: Valid values are:
* server_name - Default target is `server`. If no target is specified then log attributes are set for the server. * cluster_name - The name of a target cluster. * instance_name - The name of a target server instance. **attribute-name**: The fully scoped name of the logging attribute. The `list-log-attributes` subcommand can be used to list the names of all currently defined attributes. **attribute-value**: The value to apply to the specified attribute.
The attribute values that can be specified include the following. Refer to "administering-the-logging-service" in Payara Server General Administration section for complete explanations of each of these values.
`com.sun.enterprise.server.logging.GFFileHandler.alarms`;; Default is `false`.
`com.sun.enterprise.server.logging.GFFileHandler.excludeFields`;;
Default is an empty string.

`com.sun.enterprise.server.logging.GFFileHandler.file`;; Default is `${com.sun.aas.instanceRoot}/logs/server.log`.
`com.sun.enterprise.server.logging.GFFileHandler.flushFrequency`;; Default is `1`.
`com.sun.enterprise.server.logging.GFFileHandler.formatter`;; Default is `com.sun.enterprise.server.logging.UniformLogFormatter`.
`com.sun.enterprise.server.logging.GFFileHandler.logtoConsole`;; Default is `false`.
`com.sun.enterprise.server.logging.GFFileHandler.maxHistoryFiles`;; Default is `0`.
`com.sun.enterprise.server.logging.GFFileHandler.multiLineMode`;; Default is `true`.
`com.sun.enterprise.server.logging.GFFileHandler.retainErrorsStastics ForHours`;; Default is `0`.
`com.sun.enterprise.server.logging.GFFileHandler.rotationLimitInBytes`;; Default is `2000000`.
`com.sun.enterprise.server.logging.GFFileHandler.rotationTimelimitInMinutes`;; Default is `0`.
`com.sun.enterprise.server.logging.SyslogHandler.useSystemLogging`;; Default is `false`. `handlers`;; Default is `java.util.logging.ConsoleHandler`.
`java.util.logging.ConsoleHandler.formatter`;; Default is `com.sun.enterprise.server.logging.UniformLogFormatter`.
`java.util.logging.FileHandler.count`;; Default is `1`.
`java.util.logging.FileHandler.formatter`;; Default is `java.util.logging.XMLFormatter`.
`java.util.logging.FileHandler.limit`;; Default is `50000`.
`java.util.logging.FileHandler.pattern`;; Default is `%h/java%u.log`.
`log4j.logger.org.hibernate.validator.util.Version`;; Default is `warn`.

Examples

Example 1 Setting the Maximum Number of Log History Files to Maintain

This example sets to 8 the maximum number of log history files for the server as a whole.

```
asadmin> set-log-attributes --target=server \
com.sun.enterprise.server.logging.GFFileHandler.maxHistoryFiles=8
com.sun.enterprise.server.logging.GFFileHandler.maxHistoryFiles logging
attribute set with value 8.
These logging attributes are set for server.
Command set-log-attributes executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- collect-log-files,
- list-log-attributes,
- list-log-levels,
- rotate-log,
- set-log-levels
- "administering-the-logging-service" in Payara Server General Administration section

Set-Log-File-Format

Sets the formatter to be used for the server log file.

Synopsis

```
asadmin [asadmin-options] set-log-file-format [--help]
[--target target]
[formatter]
```

Description

The `set-log-file-format` subcommand sets the formatter to be used for the server log file for the specified target.

Payara Server provides the ULF (UniformLogFormatter) and ODL (Oracle Diagnostics

Logging) formatters. A custom formatter can also be used by specifying the fully qualified name of a class that extends the`java.util.logging.Formatter` class.

The custom formatter class can be packaged in a JAR file and placed in domain `/lib/ext`, or implemented as an HK2 service and the JAR containing the formatter implementation installed in `glassfish/modules`.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: Specifies the target for which the formatter is being set. Specify the name of the server domain, instance, or cluster. If this option is omitted, logger attributes are set for the default server.

Operands

formatter: Specifies the fomatter to be used for the server log file. Valid values are ULF, ODL, or the fully qualified name of the custom formatter class to be used to render the log files. The default value is ODL.

Examples

Example 1 Setting the Log File Format

This example sets the server log file format to `ULF` for `server1`.

```
asadmin> set-log-file-format --target server1 ULF
Command set-log-file-format executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,

- list-loggers,
- administering-the-logging-service in the Payara Server General Administration section

Set-Log-Levels

Sets the log level for one or more loggers.

Synopsis

```
asadmin [asadmin-options] set-log-levels [--help]
[--target=target]
logger-name=logger-level[:logger-name=logger-level]*
```

Description

The `set-log-levels` subcommand sets the log level for one or more loggers. Changes take effect dynamically. Depending on the log levels you set, a server restart may be necessary.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: The server instance or cluster for which log levels will be set. Valid values are:
* `server` - The default server instance. If no target is specified then log levels are set for the default server instance. * cluster-name - The name of a cluster. * instance-name - The name of a standalone server instance.

Operands

logger-name: The name of the logger. The `list-log-levels` subcommand can be used to list the names of the current loggers. **logger-level**: The level to set for the logger. Log level values are `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, and `FINEST`. The default setting is `INFO`.

Examples

Example 1 Setting a Log Level for a Logger

This example sets the log level of the web container logger to `WARNING`.

```
asadmin> set-log-levels jakarta.enterprise.system.container.web=WARNING
Command set-log-level executed successfully.
```

Example 2 Setting the Log Level for Multiple Loggers

This example sets the log level of the web container logger to `FINE` and the log level of the EJB container logger to `SEVERE`:

```
asadmin set-log-levels jakarta.enterprise.system.container.web=FINE:
jakarta.enterprise.system.container.ejb=SEVERE
Command set-log-level executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- collect-log-files,
- list-log-attributes,
- list-log-levels,
- rotate-log,
- set-log-attributes,
- administering-the-logging-service in the Payara Server General Administration section.

Set-Log-Notifier-Configuration

Configures the settings of the Log notifier

Synopsis

```
asadmin [asadmin-options] set-log-notifier-configuration [--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[--useseparatelogfile={false|true}]
```

Description

This command can be used to configure the behaviour of the log notifier in Payara Server.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the log notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--enabled`**: Defaults to `false`. Enables or disables the notifier **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart.

+ Otherwise, a restart is required. **`--filter`**: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE) **`--useseparatelogfile`**: Defaults to `false`. Whether to use a separate log file to store notifications.

Examples

Example 1 Enable the Log Notifier and Configure Filter level

```
asadmin> set-log-notifier-configuration --enabled=true --dynamic=true --filter=INFO
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-log-notifier-configuration,

Set-Metrics-Configuration

Modifies the configuration settings of the MicroProfile Metrics service.

Synopsis

```
asadmin [asadmin-options] set-metrics-configuration [--help]
[--enabled={true|false}]
[--securityEnabled={true|false}]
[--roles=<role-list>]
[--dynamic={true|false}]
[--endpoint=<context-root[default:metrics]>]
[--target=<instance-name[default:server]>]
[--virtualservers=<virtualserver-list>]
```

Description

The `set-metrics-configuration` provides a way to set the configuration of the metrics service of the targeted config.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--enabled`**: Defaults to `true`. Enables or disables the Metrics service.

`--securityEnabled`: Defaults to `false`. Specifies whether secure access to the metrics endpoint is enabled or disabled.

`--roles`: Defaults to `microprofile`. If defined, the endpoint will be assigned to a list of the role specified as a comma-separated.

`--dynamic`: Defaults to `true`. If set to `true`, applies the changes instantly without a restart.

+ Otherwise, a restart is required.

`--endpoint`: Defaults to `metrics`. Specifies the context root used to expose the metrics endpoint.

`--target`: Specifies the target for its respective configuration.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

`--virtualservers`: If defined, the endpoint will be assigned to a list of virtual servers specified as a comma-separated list of names.

+ Otherwise, the endpoint will be assigned to all virtual servers available.

Example

Example 1 Setting the configuration of the Metrics service

This example sets the metrics service configuration for the specified target `instance1`.

```
asadmin set-metrics-configuration --enabled=true --securityEnabled=true --dynamic
=true --target instance1
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-metrics-configuration

Set-Microprofile-Healthcheck-Configuration

Configures the settings of the MicroProfile HealthCheck service.

Synopsis

```
asadmin [asadmin-options] set-microprofile-healthcheck-configuration [--help]
```

```
[--enabled={true|false}]
[--securityEnabled={true|false}]
[--roles=<role-list>]
[--endpoint=<context-root[default:health]>]
[--target=<target[default:server]>]
[--virtualservers=<virtualserver-list>]
```

Description

The `set-microprofile-healthcheck-configuration` enables or disables the MicroProfile healthcheck service and configures the endpoint where all registered health checks are exposed.

IMPORTANT	Changes made with this command will only be applied when the server or instance is restarted.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--enabled`**: Defaults to true. Enables or disables the service.

`--securityenabled`: Defaults to false. Whether to secure access to the health-check endpoint is enabled.

`--roles`: Defaults to `microprofile`. If defined, the endpoint will be assigned to a list of the role specified as a comma-separated.

`--endpoint`: Defaults to health. The context root used to expose the health checks endpoint.

`--target`: Specifies the target of this command.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

`--virtualservers`: If defined, the `/health`` endpoint will be assigned to the list of virtual servers specified as a comma-separated list of names.

+ Otherwise, the endpoint will be assigned to all virtual servers available.

Example

Example 1 Configure the MicroProfile HealthCheck service

This example configures MicroProfile Health Check with security enabled, specifies the roles allowed to access the endpoint, sets a custom endpoint context root, and associates it with the specified virtual servers:

```
asadmin set-microprofile-healthcheck-configuration \
    --enabled=true \
    --securityEnabled=true \
    --roles=ADMIN \
    --endpoint=myhealth \
    --target=server \
    --virtualservers=myapp
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-microprofile-healthcheck-configuration

Set-Monitoring-Level

This command can be used to set the monitoring level of a module.

Synopsis

```
asadmin [asadmin-options] set-monitoring-level
[--help]
[--target target]
[--module module]
[--level={HIGH|LOW|OFF}]
```

Description

This command can be used to set the monitoring level (`HIGH`/`LOW`) of a module. Monitoring can also be disabled by setting the level to `OFF`.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the monitoring level of a server module. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--module`**: The name of the module. Mandatory value.

+ NOTE: Multiple module names can be set in a comma separated string. **`--level`**: Allows you to change monitoring level for a module to `OFF`, `LOW` or `HIGH`.

Examples

Example 1 Configure the monitoring level of the JVM module to `HIGH`

```
asadmin set-monitoring-level --module=jvm --level=HIGH --target=server
```

Example 2 Configure the monitoring level of both the JVM and HTTP Service modules to `LOW`

```
asadmin set-monitoring-level --module=jvm,http-service --level=LOW --target=server
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Monitoring Basics",
- "`set-monitoring-service-configuration`",
- "`enable-monitoring`",
- "`disable-monitoring`"

Set-Monitoring-Service-Configuration

Configures the monitoring service.

Synopsis

```
asadmin [asadmin-options] set-monitoring-service-configuration
[--help]
[--target target]
[--enabled={false|true}]
[--mbeansenabled={false|true}]
[--dtraceenabled={false|true}]
```

Description

This command can be used to configure the basic settings of the Monitoring service.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the monitoring service. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--enabled`**: Defaults to `false`. Enables or disables the monitoring service **`--mbeansenabled`**: Defaults to `false`. Whether to enable or disable MBean monitoring. **`--dtraceenabled`**: Defaults to `false`. Whether to enable or disable the DTrace monitoring.

Examples

Example 1 Basic configuration of the monitoring service

```
asadmin set-monitoring-service-configuration --enabled=true --mbeansenabled=true
--target=server
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Monitoring Basics"
- "`enable-monitoring`"
- "`disable-monitoring`",
- "`set-monitoring-level`"

Set-Network-Listener-Configuration

Helps to modify an existing network listener.

Synopsis

```
asadmin [asadmin-options] set-network-listener-configuration [--help]
[--enabled={true|false}]
[--dynamic={}true|false}]
[--address=address]
[--listenerport=port]
[--listenerportrange=portrange]
[--threadpool=threadpool]
[--protocol=protocol]
[--transport=transport]
[--jkenabled=true|false]
[--target target]
listener-name
```

Description

The `set-network-listener-configuration` subcommand modifies an existing network listener.

NOTE	If you edit the `admin-listener` network listener, you must restart the server for the changes to take effect. The Administration Console will not tell you that a restart is required in this case.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: `-?`: Displays the help text for the subcommand.

`--address`: The IP address or the hostname (resolvable by DNS).

`--listenerport`: The port on which the network listener is going to be listening.

`--listenerPortRange`: Port range which the network listener can bind to.

`--threadpool`: The name of the thread pool for this listener. Specifying a thread pool is optional. The default is `http-thread-pool`.

`--protocol`: The name of the protocol for this listener.

`--transport`: The name of the transport for this listener. Specifying a transport is optional. The default value is `tcp`.

`--enabled`: If set to `true`, the default, the listener is enabled at runtime.

`--jkenabled`: If enabled, the network listener will become an Apache `mod-jk` listener. The default value is `false`.

`--target`: Creates the network listener only on the specified target. Valid values are as follows:
`server`;; Creates the network listener on the default server instance. This is the default value. `configuration-name`;; Creates the network listener in the specified configuration. `cluster-name`;; Creates the network listener on all server instances in the specified cluster. `standalone-instance-name`;; Creates the network listener on the specified standalone server instance.

Operands

name: The name of the network listener.

Examples

Example 1 Modifying a Network Listener

The following command modifies a network listener named `sampleListener`:

```
asadmin set-network-listener-configuration --dynamic true --enabled=false
--listenerport 8080 sampleListener

Command set-network-listener-configuration executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-http-listener,
- restart-http-listeners,
- create-protocol,
- create-threadpool,
- create-transport,
- create-network-listener,
- delete-network-listener,
- list-network-listeners

Set-Newrelic-Notifier-Configuration

Configures the settings of the New Relic notifier.

Synopsis

```
asadmin [asadmin-options] set-newrelic-notifier-configuration [--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[--accountId accountId]
[--key key]
```

Description

This command can be used to configure the behaviour of the New Relic notifier in Payara Server.

IMPORTANT	This command is available as long as the New Relic notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server

> Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the New Relic notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--enabled`**: Defaults to `false`. Enables or disables the log notifier **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart.

+ Otherwise, a restart is required. **`--filter`**: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE) **`--accountId`**: Defines the New Relic account Id. **`--key`**: Defines the license key to be used by the notifier.

Examples

Example 1 Enable the New Relic and set its license and account configuration

```
asadmin set-newrelic-notifier-configuration --dynamic=true --enabled=true
--accountId=1658989 --key=b5815wdxj6lF_tmMBljQa5y1603JTiLh
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-newrelic-notifier-configuration

Set-Notification-Configuration

Configures the notification service along with the log notifier.

Synopsis

```
asadmin [asadmin-options] set-notification-configuration
[--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--notifierEnabled={false|true}]
[--notifierDynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[--useseparatelogfile={false|true}]
```

Description

This command can be used to set the configuration of the Notification Service and the Log Notifier at the same time.

It functions as a combination of the `notification-configure` and `set-log-notifier-configuration` Asadmin commands.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the notification service. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--enabled`**: Defaults to `false`. Enables or disables the notification service **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart. **`--notifierEnabled`**: Defaults to `false`. Enables or disables the log notifier **`--notifierDynamic`**: Defaults to `false`. When set to `true`, applies the changes to the log notifier without a server restart.

+ Otherwise, a restart is required. **`--filter`**: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE)
`--useseparatelogfile`: Defaults to `false`. Whether to use a separate log file to store notifications.

Examples

Example 1 Enables the notification service

```
asadmin set-notification-configuration --enabled=true --dynamic=true
--notifierEnabled=true --notifierDynamic=true --useSeparateLogfile=false
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- notification-configure,
- set-log-notifier-configuration

Set-Openapi-Configuration

Configures the settings of the MicroProfile OpenAPI service.

Synopsis

```
asadmin [asadmin-options] set-openapi-configuration [--help]
[--enabled={true|false}]
[--corsHeaders={true|false}]
[--securityEnabled={true|false}]
[--roles=<role-list>]
[--endpoint=<context-root[default:openapi]>]
[--target=<target[default:server]>]
[--virtualservers=<virtualserver-list>]
```

Description

This command enables or disables the OpenAPI service.

NOTE

When the OpenAPI service is disabled, the `/openapi` endpoint will always return a `403` error and any applications deployed during this period will **not** have an OpenAPI document built.
Enabling the service again will not cause a documents to be built for any currently deployed applications.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--enabled`**: Whether the service should be enabled/disabled. Defaults to `true`.

`--corsHeaders`: Defaults to `false`. Whether CORS protection headers (e.g. `Access-Control-Allow-Origin=*`) should be added to OpenAPI endpoint responses.

`--securityEnabled`: Defaults to `false`. Whether to secure access to the OpenAPI endpoint is enabled.

`--roles`: Defaults to `microprofile`.

+ If defined, the endpoint will be assigned to a list of the role specified as a comma-separated.

`--endpoint`: Defaults to `openapi`. The context root used to expose the OpenAPI endpoint.

`--target`: Specifies the target of this command.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster. configuration-name;; Specifies a named configuration.

`--virtualservers`: If defined, the `/openapi` endpoint will be assigned to the list of virtual servers specified as a comma-separated list of names.

+ Otherwise, the endpoint will be assigned to all virtual servers available.

Examples

Example 1 Configures OpenAPI service

This example configures OpenAPI service with CORS headers enabled, security enabled, a custom endpoint context root, and association with the specified virtual servers.

```
asadmin set-openapi-configuration --enabled=true --corsHeaders=true
--securityEnabled=true --roles=ADMIN --endpoint=myapi --target=server
--virtualservers=myapp
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-openapi-configuration

Set-Requesttracing-Configuration

Configures the Request Tracing service.

Synopsis

```
asadmin [asadmin-options] set-requesttracing-configuration [--help]
[--enabled={false|true}]
[--dynamic={false|true}]
[--sampleRate] rate
[--adaptiveSamplingEnabled={false|true}]
[--adaptiveSamplingTargetCount] targetCount
[--adaptiveSamplingTimeValue] value
[--adaptiveSamplingTimeUnit] timeUnit
[--applicationsOnlyEnabled={false|true}]
[--thresholdValue] value
[--thresholdUnit] timeUnit
[--sampleRateFirstEnabled={false|true}]
[--traceStoreSize] size
[--traceStoreTimeout] timeout
[--reservoirSamplingEnabled={false|true}]
[--historicTraceStoreEnabled={false|true}]
[--historicTraceStoreSize] size
[--historicTraceStoreTimeout] timeout
```

Description

This command can be used to configure several aspects of the Request Tracing service.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--enabled`**: Defaults to `false`. Enables or disables the Request Tracing service **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without

a server restart.

+ Otherwise, a restart is required. **`--sampleRate`**: The probability of a trace being sampled.

+ The rate must be a positive number between `0` and `1.0`
`--adaptiveSamplingEnabled`: Defaults to `false`. Whether to use an adaptive sampling rate. **`--adaptiveSamplingTargetCount`**: The target number of traces to sample per the configured time window.

+ Defaults to `6`. **`--adaptiveSamplingTimeValue`**: The period of time to attempt to hit the adaptive sample target count in.

+ Defaults to `1`. **`--adaptiveSamplingTimeUnit`**: The time unit for the `adaptiveSampleTimeValue` option.

+ Defaults to `MINUTES`. **`--applicationsOnlyEnabled`**: Defaults to `true`. Whether to exclusively sample traces related to applications. **`--thresholdValue`**: Sets the number of time units which trigger the tracing of a request.

+ NOTE: Only used if `adaptiveSamplingEnabled` is set to `false`.
`--thresholdUnit`: Sets the time unit to use for the threshold.

+ NOTE: Only used if `adaptiveSamplingEnabled` is set to `false`.
`--sampleRateFirstEnabled`: Defaults to `true`. Whether to perform the sampling check before or after the threshold calculation. **`--traceStoreSize`**: Sets how many request traces will be stored.

+ Defaults to `20` trace records. **`--traceStoreTimeout`**: Sets how long before Request Tracing messages are removed.

+ This field is specified in the format `%d %h %m %s` where `%` is an integer. Any of the units can also be omitted.

+ NOTE: This option is ignored when reservoir sampling is enabled.
`--reservoirSamplingEnabled`: Defaults to `false`. Whether to use reservoir

sampling, or time based sampling. `--historicTracestoreEnabled`: Defaults to `false`. When present, enables storage of the slowest sampled historical request traces in a separate trace store. `--historicTracestoreSize`: Sets how many request traces will be stored for historical purposes.

+ Defaults to `20` trace records. `--historicTracestoreTimeout`: Sets how long before Request Tracing messages are removed from the historical trace store.

+ This field is specified in the format `%d %h %m %s` where `%` is an integer. Any of the units can also be omitted.

+ NOTE: This option is ignored when reservoir sampling is enabled.

Examples

Example 1 Configure the Request Tracing Service Threshold and Historic Trace Storage

Enables the Request Tracing service with a 10 seconds threshold and to store the last 20 traces from 1 day, 2 hours and 3 minutes ago:

```
asadmin> set-requesttracing-configuration
    --enabled=true
    --thresholdValue=10
    --thresholdUnit="SECONDS"
    --dynamic=true
    --historicTraceStoreEnabled=true
    --historicTraceStoreSize=20
    --historicTraceStoreTimeout='1d 2h 3m 30s'
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-requesttracing-configuration,
- list-requesttraces

Set-Rest-Monitoring-Configuration

Configures the REST monitoring service subsystem.

Synopsis

```
asadmin [asadmin-options] set-rest-monitoring-configuration
[--help]
[--target target]
[--enabled={false|true}]
[--securityEnabled={false|true}]
[--contextRoot contextRoot]
[--applicationName name]
```

Description

This command can be used to configure the settings of the REST Monitoring subsystem.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the REST monitoring service. Valid values are: `server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified server instance. **`--enabled`**: Defaults to `false`. Enables or disables the REST monitoring service **`--securityEnabled`**: Defaults to `false`. Whether to secure access to the REST Monitoring application is enabled. **`--contextRoot`**: Defaults to `/rest-monitoring`. The context root of the REST Monitoring application. **`--applicationName`**: Defaults to `__restmonitoring`. The name of the REST Monitoring application.

Examples

Example 1 Enables REST monitoring service on the `/rest-mon` context root

```
asadmin set-rest-monitoring-configuration --enabled=true --contextRoot /rest-mon
--target=server
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- "Monitoring Basics"
- "`get-rest-monitoring-configuration`"

Set-Slack-Notifier-Configuration

Configures the settings of the Slack notifier.

Synopsis

```
asadmin [asadmin-options] set-slack-notifier-configuration [--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[--token1 token]
[--token2 token]
[--token3 token]
```

Description

This command can be used to configure the behaviour of the Slack notifier in Payara Server.

IMPORTANT	This command is available as long as the Slack notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the Slack notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;;

Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--enabled`**: Defaults to `false`. Enables or disables the log notifier **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart.

+ Otherwise, a restart is required. **`--filter`**: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE) **`--token1`**: The first token required to authenticate to the corresponding Slack channel. **`--token2`**: The second token required to authenticate to the corresponding Slack channel. **`--token3`**: The third token required to authenticate to the corresponding Slack channel.

Examples

Example 1 Enable the Slack notifier and set its required tokens

```
asadmin set-slack-notifier-configuration --enabled=true --dynamic=true --token1
=T43CHENUC --token2=B441EFQB1 --token3=Jas8P3jCZ4wK8Y2m88dCxGkn
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-slack-notifier-configuration

Set-Snmp-Notifier-Configuration

Configures the settings of the SNMP notifier.

Synopsis

```
asadmin [asadmin-options] set-snmp-notifier-configuration [--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[--hostname hostname]
[--snmpPort port]
[--community community]
[--oid oid]
[--version version]
```

Description

This command can be used to configure the behaviour of the SNMP notifier in Payara Server.

IMPORTANT	This command is available as long as the SNMP notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the SNMP notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;; Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. **`--enabled`**: Defaults to `false`. Enables or disables the log notifier **`--dynamic`**: Defaults to `false`. When set to `true`, applies the changes without a server restart.

+ Otherwise, a restart is required. **`--filter`**: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE) **`--hostName`**: Defines the hostname of the SNMP server. **`--snmpPort`**: Defaults to `162`. Defines the port number of the SNMP server. **`--version`**: Defines the version of the SNMP protocol. Will be set to a default value if not set by the user. **`--community`**: The community string used in the SNMP query. Will be set to a default value if not set by the user. **`--oid`**: The Object ID used in the SNMP query. Will be set to a default value if not set by the user.

Examples

Example 1 Enable the SNMP notifier and configure the SNMP trap settings

```
asadmin set-snmp-notifier-configuration --enabled=true --dynamic=true --hostname
```

```
=localhost --snmpPort=162 --community=public --oid=".1.3.6.1.2.1.1.8" --version=v2c
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-snmp-notifier-configuration

Set-Teams-Notifier-Configuration

Configures the settings of the MS Teams notifier.

Synopsis

```
asadmin [asadmin-options] set-teams-notifier-configuration [--help]
[--target target]
[--enabled={false|true}]
[--dynamic={false|true}]
[--filter={INFO|WARNING|SEVERE}]
[webhook-url]
```

Description

This command can be used to configure the behaviour of the MS Teams notifier in Payara Server.

IMPORTANT	This command is available as long as the Datadog notifier's core module is installed in the server. This notifier's extension is installed by default on Payara Server Enterprise.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option helps specify the target on which you are configuring the MS Teams notifier. Valid values are:
`server`;; Applies to the default server instance. This is the default value. cluster_name;;

Applies to every server instance in the cluster. instance_name;; Applies to a specified sever instance. `--enabled`: Defaults to `false`. Enables or disables the log notifier `--dynamic`: Defaults to `false`. When set to `true`, applies the changes without a server restart.

+ Otherwise, a restart is required. `--filter`: Defaults to `WARNING`. The notifier only responds to events at or above this level. (INFO, WARNING, SEVERE)

Operands

webhook-url: The url of the MS Teams channel webhook to link the notifier to.

Examples

Example 1 Enable the MS Teams notifier and set its webhook URL

```
asadmin set-teams-notifier-configuration --enabled=true --dynamic=true
https://outlook.office.com/webhook/xyz
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- get-teams-notifier-configuration

Set-Toml-Config-Source-Configuration

Configures the toml configuration source.

Synopsis

```
asadmin [asadmin-options] set-toml-config-source-configuration [--help]
[--path=<path>]
[--depth=<depth>]
[--dynamic={true|false}]
[--enabled={true|false}]
[--target=<target[default:server]>]
```

Description

The `set-toml-config-source-configuration` provides configuration options to

set the properties required by toml config source to read the values from the configured file.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--path`**: Absolute path of the TOML file or relative path to the current domain. **`--depth`**: Maximum recursion depth for nested tables/arrays. **`--dynamic`**: Configure the config source dynamically, which will not require a restart. **`--enabled`**: Enables or disables the config source. **`--target`**: Specifies the target where the configuration source resides.
Valid values are as follows:
`server`;; Specifies the DAS (default). instance-name;; Specifies a Payara Server instance. cluster-name;; Specifies a cluster.

Example

Example 1 Configuring the toml Config Source.

This example configures the toml config source.

```
asadmin set-toml-config-source-configuration --path /Users/payara/config.toml
--depth 5 --enabled true --dynamic true
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m
- get-toml-config-source-configuration

Set-Web-Context-Param

Sets a servlet context-initialization parameter of a deployed web application or module.

Synopsis

```
asadmin [asadmin-options] set-web-context-param [--help]
--name=context-param-name
{--value=value|--ignoredescriptoritem={false|true}}
[--description=description] application-name[/module]
```

Description

The `set-web-context-param` subcommand sets a servlet context-initialization parameter of one of the following items:

- A deployed web application
- A web module in a deployed Jakarta EE application

The application must already be deployed. Otherwise, an error occurs.

This subcommand enables you to change the configuration of a deployed application without the need to modify the application's deployment descriptors and repackage and redeploy the application.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--name`**: The name of the servlet context-initialization parameter that is to be set. **`--value`**: The value to which the servlet context-initialization parameter is to be set.
Either the `--value` option or the `--ignoredescriptoritem` option must be set.
`--ignoredescriptoritem`: Specifies whether the servlet context-initialization parameter is ignored if it is set in the application's deployment descriptor. When a parameter is ignored, the application behaves as if the parameter had never been set in the application's deployment descriptor. The behavior of an application in this situation depends on the application.
The possible values are as follows:
`false`;; The value is not ignored (default). `true`;; The value is ignored.

Either the `--value` option or the `--ignoredescriptoritem` option must be set.

NOTE	Do not use the `--ignoredescriptoritem` option to unset a servlet context-initialization parameter that has previously been set by using the `set-web-context-param` subcommand. Instead, use the unset-web-context-param subcommand for this purpose.

`--description`: An optional textual description of the context parameter that is being set.

Operands

application-name: The name of the application. This name can be obtained from the Administration Console or by using the list-applications subcommand.
The application must already be deployed. Otherwise, an error occurs. **module**: The relative path to the module within the application's enterprise archive (EAR) file. The path to the module is specified in the `module` element of the application's `application.xml` file.
module is required only if the servlet context-initialization parameter applies to a web module of a Jakarta EE application. If specified, module must follow application-name, separated by a slash (/).
For example, the `application.xml` file for the `myApp` application might specify the following web module:

+

```
<module>
    <web>
        <web-uri>myWebModule.war</web-uri>
    </web>
</module>
```

```
The module would be specified as the operand of this command as
`myApp/myWebModule.war`.
```

Examples

Example 1 Setting a Servlet Context-Initialization Parameter for a Web Application

This example sets the servlet context-initialization parameter `javax.faces.STATE_SAVING_METHOD` of the web application `basic-ezcomp` to `client`. The description `The location where the application's state is preserved` is provided for this parameter.

```
asadmin> set-web-context-param --name=javax.faces.STATE_SAVING_METHOD
--description="The location where the application's state is preserved"
--value=client basic-ezcomp

Command set-web-context-param executed successfully.
```

Example 2 Ignoring a Servlet Context-Initialization Parameter That Is Defined in a Deployment Descriptor

This example ignores the servlet context-initialization parameter `javax.faces.PROJECT_STAGE` of the web application `basic-ezcomp`.

```
asadmin> set-web-context-param --name=javax.faces.PROJECT_STAGE
--ignoredescriptoritem=true
basic-ezcomp
Command set-web-context-param executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- list-web-context-param,
- unset-web-context-param

Set-Web-Env-Entry

Sets an environment entry for a deployed web application or module.

Synopsis

```
asadmin [asadmin-options] set-web-env-entry [--help]
--name=env-entry-name --type=env-entry-type
{--value=value|--ignoredescriptoritem={true|false}}
```

```
[--description=description] application-name[/module]
```

Description

The `set-web-env-entry` subcommand sets an environment entry for one of the following items:

- A deployed web application
- A web module in a deployed Jakarta EE application

The application must already be deployed. Otherwise, an error occurs.

An application uses the values of environment entries to customize its behavior or presentation.

This subcommand enables you to change the configuration of a deployed application without the need to modify the application's deployment descriptors and repackage and redeploy the application.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--name`**: The name of the environment entry that is to be set. The name is a JNDI name relative to the `java:comp/env` context. The name must be unique within a deployment component. **`--type`**: The fully-qualified Java type of the environment entry value that is expected by the application's code. This type must be one of the following Java types:
* `java.lang.Boolean` * `java.lang.Byte` * `java.lang.Character` * `java.lang.Double` * `java.lang.Float` * `java.lang.Integer` * `java.lang.Long` * `java.lang.Short` * `java.lang.String` **`--value`**: The value to which the environment entry is to be set. If the `--type` is `java.lang.Character`, the value must be a single character. Otherwise, the value must be a string that is valid for the constructor of the specified type.

Either the `--value` option or the `--ignoredescriptoritem` option must be set.
`--ignoredescriptoritem`: Specifies whether the environment entry is ignored if it is set in the application's deployment descriptor. When an environment entry is ignored, the application behaves as if the entry had never been set in the application's deployment descriptor. The behavior of an application in this situation depends on the application.
The possible values are as follows:
`false`;; The value is not ignored (default). `true`;; The value is ignored.
Either the `--value` option or the `--ignoredescriptoritem` option must be set.

NOTE	Do not use the `--ignoredescriptoritem` option to unset an environment entry that has previously been set by using the `set-web-env-entry` subcommand. Instead, use the unset-web-env-entry subcommand for this purpose.

`--description`: An optional textual description of the environment entry that is being set.

Operands

application-name: The name of the application. This name can be obtained from the Administration Console or by using the list-applications subcommand.
The application must already be deployed. Otherwise, an error occurs. **module**: The relative path to the module within the application's enterprise archive (EAR) file. The path to the module is specified in the `module` element of the application's `application.xml` file.
module is required only if the environment entry applies to a web module of a Java EE application. If specified, module must follow application-name, separated by a slash (/). For example, the `application.xml` file for the `myApp` application might specify the following web module:

+

```
<module>
```

```
    <web>
        <web-uri>myWebModule.war</web-uri>
    </web>
</module>
```

The module would be specified as the operand of this command as `myApp/myWebModule.war`.

Examples

Example 1 Setting an Environment Entry for a Web Application

This example sets the environment entry `Hello User` of the application hello to `techscribe`. The Java type of this entry is `java.lang.String`.

```
asadmin> set-web-env-entry --name="Hello User"
--type=java.lang.String --value=techscribe
--description="User authentication for Hello appplication" hello
Command set-web-env-entry executed successfully.
```

Example 2 Ignoring an Environment Entry That Is Defined in a Deployment Descriptor

This example ignores the environment entry `Hello Port` of the web application `hello`.

```
asadmin> set-web-env-entry --name="Hello Port"
--type=java.lang.Integer --ignoredescriptoritem=true hello
Command set-web-env-entry executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- list-applications,
- list-web-env-entry,
- unset-web-env-entry

Set

Sets the values of configurable attributes.

Synopsis

```
asadmin [asadmin-options] set [--help]
attribute-name=value
```

Description

The `set` subcommand uses dotted names to modify the values of one or more configurable attributes.

Attributes from the monitoring hierarchy are read-only, but configuration attributes can be modified. You can use the list subcommand to display the dotted names that represent individual server components and subsystems. For example, a dotted name might be `server.applications.web-module`. After you discover the particular component or subsystem, you can then use the `get` subcommand to access the attributes. For more detailed information on dotted names, see the dotted-names help page.

NOTE

Characters that have special meaning to the shell or command interpreter, such as * (asterisk), should be quoted or escaped as appropriate to the shell, for example, by enclosing the argument in quotes. In multimode, quotes are needed only for arguments that include spaces, quotes, or backslash.

By modifying attributes, you can enable and disable services, and customize how an existing element functions. An `asadmin` subcommand is provided to update some elements. For example, `update-password-alias`.However, to update other elements, you must use the `set` command. For example, you create a JDBC connection pool by using the `create-jdbc-connection-pool` subcommand. To change attribute settings later, you use the `set` command.

Any change made by using the `asadmin` utility subcommands or the Administration Console are automatically applied to the associated Payara Server configuration file.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

attribute-name=value: Identifies the full dotted name of the attribute name and its value.

Examples

Example 1 Setting a JDBC Connection Pool Attribute

This example changes the steady pool size of the `DerbyPool` connection pool to 9.

```
asadmin> set resources.jdbc-connection-pool.DerbyPool.steady-pool-size=9
Command set executed successfully.
```

Example 2 Enabling the Monitoring Service for a Monitorable Object

This example enables monitoring for the JVM.

```
asadmin> set server.monitoring-service.module-monitoring-levels.jvm=HIGH
Command set executed successfully.
```

Example 3 Turning on Automatic Recovery for the Transaction Service

This example turns on automatic recovery for the transaction service.

```
asadmin> set server.transaction-service.automatic-recovery=true
Command set executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m

- get
- list
- dotted-names

Setup-Ssh

Sets up an SSH key on specified hosts.

Synopsis

```
asadmin [asadmin-options] setup-ssh [--help]
[--sshport ssh-port] [--sshuser ssh-user]
[--sshkeyfile ssh-keyfile] [--sshpublickeyfile ssh-public-keyfile]
[--generatekey={false|true}]
host-list
```

Description

The `setup-ssh` subcommand sets up a secure shell (SSH) key on the hosts that are specified as the operand of the subcommand. This key enables Payara Server to use public-key authentication for authentication of the user's SSH login on remote hosts.

SSH ensures that Payara Server clusters that span multiple hosts can be administered centrally. When a user runs a subcommand for cluster administration that acts on multiple hosts, the subcommand is propagated from the domain administration server (DAS) host to remote hosts. To propagate subcommands that act on a Payara Server instance that is not running, or on a node where no instances are running, Payara Server uses SSH. SSH provides confidentiality and security for data that is exchanged between the DAS and remote hosts.

Public-key authentication uses an SSH key pair that comprises the following keys:

- A private key, which is stored in a secure location on the DAS host and which may be protected with a passphrase
- The public key, which is stored on all the remote hosts with which the DAS communicates

The subcommand does not require any configuration information from the DAS and

does not modify the configuration of the DAS.

This subcommand is supported in local mode only.

Prerequisites for Using the `setup-ssh` Subcommand

To use the `setup-ssh` subcommand, the SSH user must be able to use SSH to log in to remote hosts where SSH is to be set up. Specifically, the following prerequisites must be met:

- The `ssh` client is installed on the DAS host and is accessible through the DAS user's path.
- The `sshd` daemon is installed and running on all hosts where an SSH key is to be set up.
- The user that the `--sshuser` option specifies has an SSH login on all hosts where an SSH key is to be set up.
- The `ssh-keygen` utility is installed on the DAS host either at the default location or in a location that is defined in the DAS user's path.
- On Windows systems, the SSH package for Cygwin or an MKS Software toolkit that provides SSH is installed.

Behavior of the `setup-ssh` Subcommand

The subcommand sets up SSH connectivity between the DAS host and remote hosts by automating the following tasks:

- Generating an SSH key pair. If no SSH key pair exists, the default behavior of the subcommand is to prompt the user to generate an SSH key pair. The SSH key pair is generated without an encryption passphrase. If a passphrase-protected key pair is required, the key pair must be generated manually by using the SSH command `ssh-keygen`.
- Distributing the public key. The subcommand appends the content of the public key file to the user-home`/.ssh/authorized_keys` file on each remote host. By default, the

subcommand locates the public key file in the user-home`/.ssh` directory on the host where the subcommand is run. If the user-home`/.ssh/authorized_keys` file does not exist on a host, the subcommand creates the file. user-home is the user's home directory on a host.
To distribute the public key, authentication of the user's SSH login is required. If the private key is protected by a passphrase, the passphrase is also required. By default, the subcommand prompts the user for the password and, if necessary, the passphrase. To distribute the public key without being prompted, run the subcommand as follows:

- Set the `--interactive` option of the asadmin-1m utility to `false`.
- Set the `--passwordfile` option of the `asadmin` utility to a file in which the `AS_ADMIN_SSHPASSWORD` entry specifies the SSH user's password for logging in to the specified hosts.
- If a passphrase is required, ensure that the file that `--passwordfile` option of the `asadmin` utility specifies also contains an entry for `AS_ADMIN_SSHKEYPASSPHRASE`.
  If public key authentication is already set up on a host, the subcommand informs the user that public key authentication is already set up and does not distribute the key to the host.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--sshport`**: The port to use for SSH connections to the host where SSH is being set up. The default is 22. **`--sshuser`**: The SSH user on the remote host that is to run the process for setting up SSH on that host. The default is the user that is running this subcommand. To ensure that the DAS can read this user's SSH private key file, specify the user that is running the DAS process. **`--sshkeyfile`**: The absolute path to the SSH private key file for user that the `--sshuser` option specifies. This file is used for authentication to the `sshd` daemon on the host.
The user that is running this subcommand must be able to reach the path to the key file

and read the key file.
The default is a key file in the user's `.ssh` directory on the host where the subcommand is run. If multiple key files are found, the subcommand uses the following order of preference:
. `id_rsa` . `id_dsa` . `identity` **`--sshpublickeyfile`**: The absolute path to the SSH public key file for user that the `--sshuser` option specifies. The content of the public key file is appended to the user's `.ssh/authorized_keys` file on each host where SSH is being set up. If the `.ssh/authorized_keys` file does not exist on a host, the subcommand creates the file.
The user that is running this subcommand must be able to reach the path to the key file and read the key file.
The default is a key file in the user's `.ssh` directory on the host where the subcommand is run. If multiple key files are found, the subcommand uses the following order of preference:
. `id_rsa.pub` . `id_dsa.pub` . `identity.pub` **`--generatekey`**: Specifies whether the subcommand generates the SSH key files without prompting the user.
Possible values are as follows:
`true`;; The subcommand generates the SSH key files without prompting the user.
`false`;; The behavior of the subcommand depends on whether the SSH key files exist:
* If the SSH key files exist, the subcommand does not generate the files. * If the SSH key files do not exist, the behavior of the subcommand depends on the value of the `--interactive` option of the `asadmin` utility: **If the `--interactive` option is `true`, the subcommand prompts the user to create the files.** If the `--interactive` option is `false`, the subcommand fails.
This value is the default.

Operands

host-list: A space-separated list of the names of the hosts where an SSH key is to be set up.

Examples

Example 1 Setting Up an SSH Key

This example sets up an SSH key for the user `gfuser` on the hosts `sj03` and `sj04`. The key file is not generated but is copied from the user's `.ssh` directory on the host where the subcommand is running.

```
asadmin> setup-ssh sj03 sj04
Enter SSH password for gfuser@sj03>
Copied keyfile /home/gfuser/.ssh/id_rsa.pub to gfuser@sj03
Successfully connected to gfuser@sj03 using keyfile /home/gfuser/.ssh/id_rsa
Copied keyfile /home/gfuser/.ssh/id_rsa.pub to gfuser@sj04
Successfully connected to gfuser@sj04 using keyfile /home/gfuser/.ssh/id_rsa
Command setup-ssh executed successfully.
```

Example 2 Generating and Setting Up an SSH Key

This example generates and sets up an SSH key for the user `gfuser` on the hosts `sua01` and `sua02`.

```
asadmin> setup-ssh --generatekey=true sua01 sua02
Enter SSH password for gfuser@sua01>
Created directory /home/gfuser/.ssh
/usr/bin/ssh-keygen successfully generated the identification
/home/gfuser/.ssh/id_rsa
Copied keyfile /home/gfuser/.ssh/id_rsa.pub to gfuser@sua01
Successfully connected to gfuser@sua01 using keyfile /home/gfuser/.ssh/id_rsa
Copied keyfile /home/gfuser/.ssh/id_rsa.pub to gfuser@sua02
Successfully connected to gfuser@sua02 using keyfile /home/gfuser/.ssh/id_rsa
Command setup-ssh executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- `ssh`,
- `ssh-keygen`
- `sshd`
- `Cygwin Information and Installation`
- `MKS Software`

Show-Component-Status

Displays the status of the deployed component.

Synopsis

```
asadmin [asadmin-options] show-component-status [--help]
[--target target] component-name
```

Description

The `show-component-status` subcommand gets the status (either enabled or disabled) of the deployed component.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--target`**: This option specifies the target on which you are showing the component status. Valid values are:
`server`;; Shows the component status for the default server instance `server` and is the default value. `domain`;; Shows the component status for the domain. cluster_name;; Shows the component status for the cluster. instance_name;; Shows the component status for a clustered or stand-alone server instance.

Operands

component-name: The name of the component whose status is to be listed.
The name can include an optional version identifier, which follows the name and is separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. To list multiple versions, you can use an asterisk (`*`) as a wildcard character. For more information about module and application versions, see module-and-application-versions in Payara Server Application Deployment section.

Examples

Example 1 Showing the Status of a Component

This example gets the status of the `MEjbApp` component.

```
asadmin> show-component-status MEjbApp
Status of MEjbApp is enabled
Command show-component-status executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-applications,
- list-sub-components

Start-Cluster

Starts a cluster.

Synopsis

```
asadmin [asadmin-options] start-cluster [--help]
[--verbose={false|true}] cluster-name
```

Description

The `start-cluster` subcommand starts a cluster by starting all Payara Server instances in the cluster that are not already running. This subcommand requires the secure shell (SSH) to be configured on the host where the domain administration server (DAS) is running and on all hosts where instances in the cluster reside.

NOTE	If all instances reside on the same host as the DAS, SSH is not required. You might require to start a cluster in which instances reside on hosts where SSH is not configured that are remote from the DAS. In this situation, run the start-local-instance subcommand for each instance from the host where the instance resides.

You may run this subcommand from any host that can contact the DAS.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: `-?`: Displays the help text for the subcommand.

`--verbose`: Specifies whether additional status information is displayed when the cluster is started.

+ Valid values are as follows:

+ `true`;; Displays the command to start each instance in the cluster and whether the attempt to start each instance succeeded. `false`;; Displays no additional status information (default).

Operands

cluster-name: The name of the cluster to start.

Examples

Example 1 Staring a Cluster

This example starts the cluster `ymlcluster`. Additional status information is displayed when the cluster is started.

```
asadmin> start-cluster --verbose ymlcluster
start-instance yml-i-sr1-usca-02
start-instance yml-i-sr1-usca-01

The command start-instance executed successfully for:
yml-i-sr1-usca-02 yml-i-sr1-usca-01

Command start-cluster executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-cluster,
- delete-cluster,
- list-clusters,
- setup-ssh,
- start-local-instance,
- stop-cluster,
- restart-cluster

Start-Database

Starts a standalone H2 database in server mode.

Synopsis

```
asadmin [asadmin-options] start-database [--help]
[--jvmoptions jvm-options]
[--dbhost host] [--dbport port-no]
[--dbhome db-file-path]
```

Description

The `start-database` subcommand starts the H2 database server that is available for use with Payara Server. Use this subcommand only for working with applications deployed to the server.

When you start the H2 server by using the `start-database` subcommand, the database server is started in Network Server mode. Clients connecting to it must use the corresponding H2 JDBC client driver. For details on how to connect to the database, refer to the official H2 documentation.

When the database server starts, or a client connects to it successfully, the following files are created:

- The `h2.log` file that contains the database server process log along with its standard output and standard error information

- The database files that contain your schema (for example, database tables)

These files are created at the location that is specified by the `--dbhome` option. To create the database files at a particular location, you must set the `--dbhome` option. If the `--dbhome` option is not specified, the `start-database` subcommand determines where to create these files as follows:

- If the current working directory contains a file that is named `h2.log`, the `start-database` subcommand creates the files in the current working directory.
- Otherwise, the `start-database` subcommand creates the files in the `as-install/databases` directory.

The `start-database` subcommand starts the database process, even if it cannot write to the log file.

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--jvmoptions`**: A space-separated list of command-line options that are passed to the Java application launcher when the database is started. By default, no options are passed.

+ The format of an option depends on whether the option has a name and a value or only a name:

- If the option has a name and a value, the format is option-name=value.
- If the option has only a name, the format is option-name. For example, `-Xmx512m`. **`--dbhost`**: The host name or IP address of the H2 database server process. The default is the IP address 0.0.0.0, which denotes all network interfaces on the host where you run the `start-database` subcommand. **`--dbport`**: The port number where the H2 database server listens for client connections. This port must be available for the listen socket, otherwise the database server will not start.

`--dbhome`: The absolute path to the directory where the database files and the `h2.log` file are created. If the `--dbhome` option is not specified, the `start-database` subcommand determines where to create these files as follows:

- If the current working directory contains a file that is named `h2.log`, the `start-database` subcommand creates the files in the current working directory.
- Otherwise, the `start-database` subcommand creates the files in the as-install`/databases` directory.
  To create the database files at a particular location, you must set the `--dbhome` option.

Examples

Example 1 Starting Java DB

This example starts H2 on the host `host1` and port 5001.

```
asadmin> start-database --dbhost host1 --dbport 5001 --terse=true

Starting database in Network Server mode on host 0.0.0.0 and port 9092.
tcp://10.228.1.149:9092
Starting database in the background.
Log redirected to /opt/databases/h2.log.
```

Example 2 Starting H2 with Options for the Java Application Launcher

This example starts H2 with the options for setting the minimum heap memory size to 128 megabytes and the maximum heap memory size to 512 megabytes.

```
asadmin> start-database --jvmoptions="-Xms128m -Xmx512m" --terse=true

Starting database in Network Server mode on host 0.0.0.0 and port 9092.
tcp://10.228.1.149:9092
Starting database in the background.
Log redirected to /opt/payara/glassfish/databases/h2.log.
```

Exit Status

The exit status applies to errors in executing the `asadmin` utility.For information on database errors, see the `h2.log` file. This file is located in the directory you specify by using the `--dbhome` option when you run the `start-database` subcommand. If you

did not specify `--dbhome`, the value of `H2_INSTALL` defaults to `as-install/h2`.

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- stop-database
- administering-database-connectivity in Payara Server General Administration section

Start-Deployment-Group

Starts all the instances in a Payara Server deployment group.

Synopsis

```
asadmin [asadmin-options] start-deployment-group [--help]
[--instanceTimeout timeout]
[--timeout timeout]
[--verbose={false|true}]
name
```

Description

The `start-deployment-group` subcommand starts all instances that belong to a Payara Server deployment group. The command will fail if the deployment group doesn't contain any instances in it.

Internally this command works in the same manner that the "`start-instance`" subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--verbose`: **`-v`**: Whether to start the instances in verbose mode. Defaults to `false`.

`--timeout`: Specifies how long to take for the instances to start in seconds. If starting the instances takes longer than this amount then this command will fail.

+ The default value is `600`.

`--instanceTimeout`: Specifies how long to take for each instance to start in seconds. If starting the instances takes longer than this amount then this command will fail.

+ The default value is `600`.

Operands

name: The name of the deployment group. If there are no deployment groups that match the name the command will exit with an error.

Examples

Example 1 Start a Deployment Group

This example starts a deployment group that is named `test-dgroup` that contains one instance:

```
asadmin start-deployment-group test-dgroup
50%: start-deployment-group: Executing start-instance on 1 instances.
Command start-deployment-group executed successfully.
```

Example 2 Start a Deployment Group with timeout

This example starts a deployment group that is named `test-dgroup` that contains one instance using an instance timeout of 30 seconds:

```
asadmin start-deployment-group --instanceTimeout 30 test-dgroup
50%: start-deployment-group: Executing start-instance on 1 instances.
Command start-deployment-group executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-deployment-group,
- start-instance,
- list-deployment-groups,
- stop-deployment-group,
- restart-deployment-group

Start-Domain

Starts the Domain Administration Server(DAS) of the specified domain.

Synopsis

```
asadmin [asadmin-options] start-domain [--help]
[--debug={true|false}]
[--domaindir domain-dir]
[--dry-run={true|false}]
[--verbose={true|false}]
[--watchdog={true|false}]
[--timeout timeout]
[--warmup={true|false}]
[--prebootcommandfile path-to-file]
[--postbootcommandfile path-to-file]
[domain-name]
```

Description

The `start-domain` subcommand starts the domain administration server (DAS) of the specified domain. If a domain is not specified, the default domain is assumed. If the domains directory contains two or more domains, the domain-name operand must be specified.

NOTE	On the Windows platform, processes can bind to the same port. To avoid this problem, do not start multiple domains with the same port number at the same time.

This subcommand is supported in local mode only.

NOTE	In Payara Server, the `start-domain` subcommand prompts for a new

> admin user password if no password has been set for the admin user. Additionally, the admin user password must not be blank if secure administration is enabled; otherwise, the `start-domain` subcommand fails.
> To provide the new admin user password, you can use the `--passwordfile` utility option of the asadmin-1m command after adding the entry `AS_ADMIN_NEWPASSWORD` to the password file.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--debug`: **`-d`**: Specifies whether the domain is started with Java Platform Debugger Architecture (JPDA) debugging enabled. With debugging enabled extra exceptions can be printed.

+ Possible values are as follows:

`true`

The instance is started with JPDA debugging enabled and the port number for JPDA debugging is displayed.

`false`

The instance is started with JPDA debugging disabled (default).

`--dry-run`: **`-n`**: Suppresses actual starting of the domain. Instead, `start-domain` displays the full java command that would be used to start the domain, including all options. Reviewing this command can be useful to confirm JVM options and when troubleshooting startup issues.

+ The default value is `false`.

`--domaindir`: The domain root directory, which contains the directory of the domain

that is to be restarted. If specified, the path must be accessible in the file system. The default location of the domain root directory is `as-install/domains`.

`--verbose`: **`-v`**: Specifies whether detailed information about the domain is displayed in the console window where the subcommand is run.

+ Possible values are as follows:

`true`

Detailed startup messages and log messages about the domain are displayed in the console window where the subcommand is run. If the domain is later restarted by running the restart-domain subcommand from a different console window, messages continue to be displayed in the original console window.

You can kill the Payara Server process by typing `CTRL-C` in the console window.

You can kill the Payara Server process and obtain a thread dump for the server by typing one of the following key combinations in the console window:

- `CTRL-\` on UNIX systems
- `CTRL-Break` on Windows systems

`false`

Detailed information is not displayed (default).

`--watchdog`: **`-w`**: Specifies whether limited information about the domain is displayed in the console window where the subcommand is run. The `--watchdog` option is similar to `--verbose` but does not display the detailed startup messages and log messages. This option is useful when running the `asadmin` utility in the background or with no attached console.

+ Possible values are as follows:

`true`

Limited information is displayed in the console window.

`false`

Limited information is not displayed in the console window (default).

`--prebootcommandfile`: Path to file containing commands to run before booting the server. Only a limited amount of commands will work at this point.

`--postbootcommandfile`: Path to file containing commands to run after the server's boot sequence is completed.

`--timeout`: Specifies how long to take for the domain to start in seconds. If starting the domain takes longer than this amount then this command will fail.

+ The default value is `600`.

`--warmup`: Specifies if the server will stop after setting configurations.

+ The default value is `false`.

Operands

domain-name: The unique name of the domain you want to start.

+ This operand is optional if only one domain exists in the Payara Server installation.

Examples

Example 1 Starting a Domain

This example starts `mydomain4` in the default domains directory.

```
asadmin> start-domain mydomain4
Waiting for DAS to start. ..........
Started domain: mydomain4
Domain location: /myhome/payara7/glassfish/domains/mydomain4
Log file: /myhome/payara7/glassfish/domains/mydomain4/logs/server.log
Admin port for the domain: 4848
Command start-domain executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-domain,
- delete-domain,
- list-domains,
- restart-domain,
- restart-domains,
- start-domains,
- stop-domain,
- stop-domains,
- stop-all-domains

Start-Domains

Starts the Domain Administration Server(DAS) of the specified domains.

Synopsis

```
asadmin start-domains [--help]
[--domainTimeout timeout]
[--timeout timeout]
[domain-name]
```

Description

The `start-domains` subcommand can be used to start the specified list of domains in the operand. This list must be specified as a comma separated string of values.

NOTE	The domains will be started assuming default values for all options you would specify when running the start-domain command against one domain (default domain directory, debug mode off, verbose mode off, etc.)

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--timeout`: Specifies how long to take for the domains to start in seconds. If starting the domains takes longer than this amount then this command will fail.

+ The default value is `600`.

`--domainTimeout`: Specifies how long to take for each domain to start in seconds. If starting the domains takes longer than this amount then this command will fail.

+ The default value is `600`.

Operands

domain-names: The list of unique names of the domains you want to start.

Examples

Example 1 Starting multiple Domains

This example starts `domain1` and `testy` in the default domains directory.

```
asadmin> start-domains domain1,testy
Waiting for domain1 to start ..............................
Successfully started the domain : domain1
domain  Location: /myhome/payara7/glassfish/domains/domain1
Log File: /myhome/payara7/glassfish/domains/domain1/logs/server.log
Admin Port: 4848
Started domain domain1
Waiting for testy to start ..............................
Successfully started the domain : testy
domain  Location: /myhome/payara7/glassfish/domains/testy
Log File: /myhome/payara7/glassfish/domains/domain1/logs/server.log
Admin Port: 6048
Started domain testy
Command start-domains executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-domains,
- restart-domain,
- restart-domains,
- start-domain,
- stop-domain,
- stop-domains,
- stop-all-domains

Start-Instance

Starts a Payara Server instance.

Synopsis

```
asadmin [asadmin-options] start-instance [--help]
[--debug={false|true}] [--sync={normal|full|none}]
[--timeout timeout] instance-name
```

Description

The `start-instance` subcommand starts a Payara Server instance. This subcommand requires the secure shell (SSH) to be configured on the machine where the domain administration server (DAS) is running and on the machine where the instance resides.

NOTE

SSH is not required if the instance resides on a node of type `CONFIG` that represents the local host. A node of type `CONFIG` is not enabled for remote communication over SSH.

You may run this subcommand from any machine that can contact the DAS.

The subcommand can start any Payara Server instance, regardless of how the instance was created. For example, this subcommand can start an instance that was created by using the create-local-instance subcommand.

This command is supported in remote mode only.

Options

CAUTION	These are not the same options available for the `start-local-instance` command.

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--debug`: Specifies whether the instance is started with Java Platform Debugger Architecture (JPDA) debugging enabled.

+ Possible values are as follows:

`true`

The instance is started with JPDA debugging enabled and the port number for JPDA debugging is displayed.

`false`

The instance is started with JPDA debugging disabled (default).

`--sync`: The type of synchronization between the DAS and the instance's file when the instance is started.

+ Possible values are as follows:

`none`

The DAS does not synchronize the instance's files with any changes. This type of synchronization minimizes the time that is required to start the instance.

`normal`

The DAS synchronizes the instance with changes since the last synchronization as follows:

- For the `config` directory, the DAS synchronizes the instance with all changes.
- For the `applications` directory and `docroot` directory, only a change to a top-level subdirectory causes the DAS to synchronize all files under that subdirectory.

  If a file below a top level subdirectory is changed without a change to a file in the top level subdirectory, full synchronization is required. In normal operation, files below the top level subdirectories of these directories are not changed. If an application is deployed and undeployed, full synchronization is not necessary to update the instance with the change.

  This value is the default.

`full`

The DAS synchronizes the instance with all of the instance's files, regardless of whether the files have changed since the last synchronization. This type of synchronization might delay the startup of the instance while the DAS updates all files in the instance's directories.

WARNING	Using full synchronization will cause the configuration to be deleted if the DAS cannot be reached.

`--timeout`: Specifies how long to take for the instance to start in seconds. If starting the instance takes longer than this amount then this command will fail (but the instance may still continue its startup process).

+ The default value is `120`.

`--terse`: Specifies whether information on the result is output

+ The default value is `false`.

Operands

instance-name: The name of the Payara Server instance to start.

Examples

Example 1 Starting a Payara Server Instance

This example starts the Payara Server instance `pmdsa1`.

```
asadmin start-instance pmdsa1
Waiting for the server to start ..............................
Successfully started the instance: pmdsa1
instance Location: /export/payara7/glassfish/nodes/localhost/pmdsa1
Log File: /export/payara7/glassfish/nodes/localhost/pmdsa1/logs/server.log
Admin Port: 24848
Command start-local-instance executed successfully.
The instance, pmdsa1, was started on host localhost

Command start-instance executed successfully.
```

Example 2 Starting a Payara Server Instance With JPDA Debugging Enabled

This example starts the Payara Server instance `ymlsa1` with JPDA debugging enabled.

```
asadmin start-instance --debug=true ymlsa1
Waiting for the server to start ..............................
Successfully started the instance: ymlsa1
instance Location: /export/payara7/glassfish/nodes/localhost/ymlsa1
Log File: /export/payara7/glassfish/nodes/localhost/ymlsa1/logs/server.log
Admin Port: 24849
Debugging is enabled. The debugging port is: 29010
Command start-local-instance executed successfully.
The instance, ymlsa1, was started on host localhost

Command start-instance executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-instance,
- create-local-instance,
- delete-instance,
- delete-local-instance,

- setup-ssh,
- start-domain,
- start-local-instance,
- stop-domain,
- stop-instance,
- stop-local-instance,
- http://www.oracle.com/technetwork/java/javase/tech/jpda-141715.html [Java Platform Debugger Architecture (JPDA)]

Start-Local-Instance

Starts a Payara Server instance on the host where the subcommand is run.

Synopsis

```
asadmin [asadmin-options] start-local-instance [--help]
[--nodedir node-dir]
[--node node]
[--debug={false|true}]
[--dry-run={true|false}]
[--sync={normal|full|none}]
[--verbose={false|true}]
[--watchdog={true|false}]
[--timeout timeout]
[instance-name]
```

Description

The `start-local-instance` subcommand starts a Payara Server instance on the host where the subcommand is run. This subcommand does not require the secure shell (SSH) to be configured. You must run this command from the host where the instance resides.

The subcommand can start any Payara Server instance, regardless of how the instance was created. For example, this subcommand can start an instance that was created by using the create-instance subcommand.

The `start-local-instance` subcommand does not contact the domain administration server (DAS) to determine the node on which the instance resides. To

determine the node on which the instance resides, the subcommand searches the directory that contains the node directories. If multiple node directories exist, the node must be specified as an option of the subcommand.

This subcommand is supported in local mode. However, to synchronize the instance with the DAS, this subcommand must be run in remote mode.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

NOTE	These are not the same as the options available for the start-instance command.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--nodedir`: Specifies the directory that contains the instance's node directory. The instance's files are stored in the instance's node directory.

+ The default is `as_install/nodes/`.

`--node`: Specifies the node on which the instance resides. This option may be omitted only if the directory that the `--nodedir` option specifies contains only one node directory. Otherwise, this option is required.

`--debug`: **`-d`**: Specifies whether the instance is started with Java Platform Debugger Architecture (JPDA) debugging enabled.

+ Possible values are as follows:

+ `true`;; The instance is started with JPDA debugging enabled and the port number for JPDA debugging is displayed. `false`;; The instance is started with JPDA debugging disabled (default).

`--dry-run`: **`-n`**: Suppresses actual starting of the instance. Instead, `start-local-instance` displays the full command that would be used to start the instance,

including all options. Reviewing this command can be useful when troubleshooting startup issues.

+ The default value is `false`.

`--sync`: The type of synchronization between the DAS and the instance's files when the instance is started.

+ Possible values are as follows:

+ `none`;; The DAS does not synchronize the instance's files with any changes. This type of synchronization minimizes the time that is required to start the instance. `normal`;; The DAS synchronizes the instance with changes since the last synchronization as follows:
* For the `config` directory, the DAS synchronizes the instance with all changes. * For the `applications` directory and `docroot` directory, only a change to a top-level subdirectory causes the DAS to synchronize all files under that subdirectory.

+ If a file below a top level subdirectory is changed without a change to a file in the top level subdirectory, full synchronization is required. In normal operation, files below the top level subdirectories of these directories are not changed. If an application is deployed and un-deployed, full synchronization is not necessary to update the instance with the change.

+ This value is the default.

`full`

The DAS synchronizes the instance with all of the instance's files, regardless of whether the files have changed since the last synchronization. This type of synchronization might delay the startup of the instance while the DAS updates all files in the instance's directories.

CAUTION	If the DAS is not running or is unreachable from the host where you are running this subcommand, do not set the `--sync`

> option to `full`. To perform a full synchronization, the subcommand removes the instance's cache. If the DAS cannot be contacted to replace the cache, the subcommand fails and the instance cannot be restarted until it is resynchronized with the DAS.

`--verbose`: **`-v`**: Specifies that the server log is printed to the console from where the subcommand is run. Control is not returned to the console after the instance has started. The instance can be stopped and control returned by inputting CTRL-C.

+ Possible values are as follows:

+ `true`;; Detailed startup messages and log messages about the instance are displayed in the console window where the subcommand is run. If the instance is later restarted by running the restart-local-instance subcommand from a different console window, messages continue to be displayed in the original console window.

+ You can kill the Payara Server process by typing `CTRL-C` in the console window.

+ You can kill the Payara Server process and obtain a thread dump for the server by typing one of the following key combinations in the console window:

+ * `CTRL-\` on UNIX systems * `CTRL-Break` on Windows systems `false`;; Detailed information is not displayed (default).

`--watchdog`: **`-w`**: Specifies whether limited information about the instance is displayed in the console window where the subcommand is run. The `--watchdog` option is similar to `--verbose` but does not display the detailed startup messages and log messages. This option is useful when running the `asadmin` utility in the background or with no attached console.

+ Possible values are as follows:

+ `true`;; Limited information is displayed in the console window. `false`;; Limited information is not displayed in the console window (default).

`--timeout`: Specifies how long to take for the local instance to start in seconds. If starting the instance takes longer than this amount then this command will fail (but the instance may still continue its startup process)

+ The default value is `600`.

Operands

instance-name: The name of the instance to start.

Examples

Example 1 Starting an Instance Locally

This example starts the instance `yml-i-sj01` on the host where the subcommand is run.

```
asadmin start-local-instance --node sj01 yml-i-sj01
Waiting for the server to start ..............................
Successfully started the instance: yml-i-sj01
instance Location: /export/payara7/glassfish/nodes/sj01/yml-i-sj01
Log File: /export/payara7/glassfish/nodes/sj01/yml-i-sj01/logs/server.log
Admin Port: 24849
Command start-local-instance executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-instance,
- create-local-instance,
- delete-instance,
- delete-local-instance,
- restart-instance,
- restart-local-instance,

- start-domain,
- start-instance,
- stop-domain,
- stop-instance,
- stop-local-instance
- Java Platform Debugger Architecture (JPDA)

Stop-All-Domains

Stops the Domain Administration Server(DAS) of all current running domains on the server's installation.

Synopsis

```
asadmin stop-all-domains [--help]
[--force={true|false}]
[--kill={true|false}]
[--timeout timeout]
[--domainTimeout timeout]
```

Description

The `stop-all-domains` subcommand can be used to stop ALL current running domains on the server's installation.

Options

`--force`: Specifies whether each domain is forcibly stopped immediately.

+ The default value is `true`.

`--kill`: Specifies whether the domain is killed by using functionality of the operating system to terminate the domain process.

+ The default value is `false`.

`--timeout`: Specifies how long to take for the domains to stop in seconds. If stopping the domains takes longer than this amount then this command will fail.

+ The default value is `600`.

`--domainTimeout`: Specifies how long to take for each domain to stop in seconds. If stopping the domain takes longer than this amount then this command will fail

+ The default value is `600`.

Examples

Example 1 Stopping Domains

This example stops all currently running domains on the server's installation.

```
asadmin stop-all-domains

Stopping domain domain1
Stopped domain
Stopping domain production
Stopped domain
Stopping domain test
Stopped domain
Stopping domain testy
Stopped domain
Command stop-all-domains executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-domain,
- delete-domain,
- list-domains,
- restart-domain,
- restart-domains,
- start-domain,
- start-domains,

- stop-domain,
- stop-domains

Stop-Cluster

Stops a Payara Server cluster.

Synopsis

```
asadmin [asadmin-options] stop-cluster [--help]
[--verbose={false|true}]
[--kill={false|true}]
cluster-name
```

Description

The `stop-cluster` subcommand stops a Payara Server cluster by stopping all running Payara Server instances in the cluster.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

`--verbose`: Specifies whether additional progress messages about the status of instances in the cluster are displayed while the cluster is being stopped.

+ Possible values are as follows:

+ `true`;; Additional progress messages about the status of instances in the cluster are displayed. `false`;; No messages about the status of instances in the cluster are displayed.

`--kill`: Specifies whether each instance in the cluster is killed by using functionality of the operating system to terminate the instance process.

+ Possible values are as follows:

+ `false`;; No instances are killed. The subcommand uses functionality of the Java platform to terminate each instance process (default). `true`;; Each instance is killed. The subcommand uses functionality of the operating system to terminate each instance process.

Operands

cluster-name: The name of the cluster to stop.

Examples

Example 1 Stopping a Cluster

This example stops the cluster `pmdcluster`. Additional progress messages about the status of instances in the cluster are displayed while the cluster is being stopped.

```
asadmin> stop-cluster --verbose pmdcluster
stop-instance pmd-i-sj01
stop-instance pmd-i-sj02

The command stop-instance executed successfully for: pmd-i-sj01 pmd-i-sj02

Command stop-cluster executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-cluster,
- delete-cluster,
- list-clusters,
- start-cluster,
- restart-cluster

Stop-Database

Stops a running H2 database server.

Synopsis

```
asadmin [asadmin-options] stop-database [--help]
[--dbuser db-user]
[--dbhost host] [--dbport port-no]
```

Description

The `stop-database` subcommand stops a running H2 database server process. The database is typically started with the start-database subcommand. A single host can have multiple database server processes running on different ports. The `stop-database` subcommand stops the database server process for the specified port only.

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--dbuser`**: The username of the Java DB user that is to stop the server process. If this option is omitted, no user is specified. **`--dbhost`**: The host name or IP address of the Java DB server process. The default is the IP address 0.0.0.0, which denotes all network interfaces on the host where you run the `stop-database` subcommand. **`--dbport`**: The port number where the H2 database server listens for client connections.

Examples

Example 1 Stopping Java DB

This example stops Java DB on host host1 and port 5001.

```
asadmin> stop-database --dbhost host1 --dbport 5001

Command stop-database executed successfully.
```

Exit Status

The exit status applies to errors in executing the `asadmin` utility. For information on database errors, see the `h2.log` file. This file is located in the directory you specify by using the `--dbhome` option when you run the `start-database` subcommand.

If you did not specify `--dbhome`, the value of `H2_INSTALL` defaults to `as-install/h2`.

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- start-database "administering-database-connectivity" in the Payara Server General Administration section

Stop-Deployment-Group

Stops all the instances in a Payara Server deployment group.

Synopsis

```
asadmin [asadmin-options] stop-deployment-group [--help]
[--instanceTimeout timeout]
[--timeout timeout]
[--kill={false|true}]
[--verbose={false|true}]
name
```

Description

The `sop-deployment-group` subcommand sop all instances that belong to a Payara Server deployment group. Initially the deployment group contains no Payara Server instances, applications, or resources.

Internally this command works in the same manner that the "`stop-instance`" subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page.

`--help`: **`-?`**: Displays the help text for the subcommand.

`--verbose`: `-v`: Whether to stop the instances in verbose mode. Defaults to `false`.

`--timeout`: Specifies how long to take for the instances to stop in seconds. If starting the instances takes longer than this amount then this command will fail.

+ The default value is `600`.

`--instanceTimeout`: Specifies how long to take for each instance to stop in seconds. If stopping the instances takes longer than this amount then this command will fail.

+ The default value is `600`. `--kill`: Whether to kill the instance processes if they do not stop gracefully. Defaults to `false`.

Operands

name: The name of the deployment group. If there are no deployment groups that match the name the command will exit with an error.

Examples

Example 1 Stop a Deployment Group

This example stops a deployment group that is named `test-dgroup` that contains one instance:

```
asadmin stop-deployment-group test-dgroup
50%: stop-deployment-group: Executing stop-instance on 1 instances.
Command stop-deployment-group executed successfully.
```

Example 2 Stop a Deployment Group with timeout

This example stops a deployment group that is named `test-dgroup` that contains one instance, using an instance timeout of 30 seconds:

```
asadmin stop-deployment-group --instanceTimeout 30 test-dgroup
50%: stop-deployment-group: Executing stop-instance on 1 instances.
Command stop-deployment-group executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m
- create-deployment-group,
- start-instance,
- list-deployment-groups,
- stop-deployment-group,
- restart-deployment-group

Stop-Domain

Stops the Domain Administration Server(DAS) of the specified domain.

Synopsis

```
asadmin [asadmin-options] stop-domain [--help]
[--domaindir domaindir]
[--force={true|false}]
[--kill={false|true}]
[--timeout timeout]
[domain-name]
```

Description

The `stop-domain` subcommand stops the Domain Administration Server (DAS) of the specified domain. If the domain directory is not specified, the domain in the default domains directory is stopped. If there are two or more domains in the domains directory, the domain-name operand must be specified.

This subcommand is supported in local or remote mode. If you specify a host name, the subcommand assumes you are operating in remote mode, which means you must correctly authenticate to the remote server. In local mode, you normally do not need to authenticate to the server as long as you are running the subcommand as the same user who started the server.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the

subcommand. `--domaindir`: Specifies the directory of the domain that is to be stopped. If specified, the path must be accessible in the file system. If not specified, the domain in the default `as-install/domains` directory is stopped. `--force`: Specifies whether the domain is forcibly stopped immediately.
Possible values are as follows:
`true`;; The domain is forcibly stopped immediately (default). `false`;; The subcommand waits until all threads that are associated with the domain are exited before stopping the domain. `--kill`: Specifies whether the domain is killed by using functionality of the operating system to terminate the domain process.
Possible values are as follows:
`false`;; The domain is not killed. The subcommand uses functionality of the Java platform to terminate the domain process (default). `true`;; The domain is killed. The subcommand uses functionality of the operating system to terminate the domain process. `--timeout`: Specifies how long to take for the domain to stop in seconds. If stopping the domain takes longer than this amount then this command will fail.

+ The default value is `600`.

Operands

domain-name: The name of the domain you want to stop. Default is the name specified during installation, usually `domain1`.

Examples

Example 1 Stopping a Domain

This example stops the domain named `sampleDomain` in the default domains directory.

```
asadmin> stop-domain sampleDomain
Waiting for the domain to stop ..................
Command stop-domain executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-domain,
- delete-domain,
- list-domains,
- restart-domain,
- restart-domains,
- start-domain,
- start-domains,
- stop-domains,
- stop-all-domains

Stop-Domains

Stops the Domain Administration Server(DAS) of the specified domains.

Synopsis

```
asadmin stop-domains [--help]
[--domainTimeout timeout]
[--timeout timeout]
[domain-name]
```

Description

The `stop-domains` subcommand can be used to stop the specified list of domains in the operand. This list must be specified as a comma separated string of values.

Options

`--timeout`: Specifies how long to take for the domains to stop in seconds. If stopping the domains takes longer than this amount then this command will fail.

+ The default value is `600`.

`--domainTimeout`: Specifies how long to take each domain to stop in seconds. If

stopping the domain takes longer than this amount then this command will fail.

+ The default value is `600`.

Operands

domain-names: The names of the domains you want to stop.

Examples

Example 1 Stopping Domains

This example stops the domains named `domain1` and `testy` in the default domains directory.

```
asadmin stop-domains domain1,testy

Stopping domain domain1
Stopped domain

Stopping domain testy
Stopped domain

Command stop-domains executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-domain,
- delete-domain,
- list-domains,
- restart-domain,
- restart-domains,
- start-domain,
- start-domains,

- stop-domain,
- stop-all-domains

Stop-Instance

Stops a running Payara Server instance.

Synopsis

```
asadmin [asadmin-options] stop-instance [--help]
[--force={false|true}]
[--kill={false|true}]
instance-name
```

Description

The `stop-instance` subcommand stops a running Payara Server instance.

The subcommand can stop any Payara Server instance, regardless of how the instance was created. For example, this subcommand can stop an instance that was created by using the create-local-instance subcommand.

This command is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--force`**: Specifies whether the instance is forcibly stopped immediately.
Possible values are as follows:
`true`;; The instance is forcibly stopped immediately (default). `false`;; The subcommand waits until all threads that are associated with the instance are exited before stopping the instance. **`--kill`**: Specifies whether the instance is killed by using functionality of the operating system to terminate the instance process.
Possible values are as follows:
`false`;; The instance is not killed. The subcommand uses functionality of the Java platform to terminate the instance process (default). `true`;; The instance is killed. The subcommand uses functionality of the operating system to terminate the instance

process.

Operands

instance-name: This is the name of the Payara Server instance to stop.

Examples

This example stops the Payara Server instance `yml-i-sj01`.

Example 1 Stopping a Payara Server Instance

```
asadmin> stop-instance yml-i-sj01
The instance, yml-i-sj01, was stopped.
Command stop-instance executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-instance,
- create-local-instance,
- delete-instance,
- delete-local-instance,
- start-domain,
- start-instance,
- start-local-instance,
- stop-domain,
- stop-local-instance

Stop-Local-Instance

Stops a Payara Server instance on the machine where the subcommand is run.

Synopsis

```
asadmin [asadmin-options] stop-local-instance [--help]
[--nodedir node-dir] [--node node]
[--force={true|false}] [--kill={false|true}]
[instance-name]
```

Description

The `stop-local-instance` subcommand stops a Payara Server instance on the machine where the subcommand is run. This subcommand does not require secure shell (SSH) to be configured. You must run this command from the machine where the instance resides.

The subcommand can stop any Payara Server instance, regardless of how the instance was created. For example, this subcommand can stop an instance that was created by using the create-instance subcommand.

The `stop-local-instance` subcommand does not contact the DAS to determine the node on which the instance resides. To determine the node on which the instance resides, the subcommand searches the directory that contains the node directories. If multiple node directories exist, the node must be specified as an option of the subcommand.

This subcommand is supported in local mode.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--nodedir`**: Specifies the directory that contains the instance's node directory. The instance's files are stored in the instance's node directory. The default is as-install`/nodes`. **`--node`**: Specifies the node on which the instance resides. This option may be omitted only if the directory that the `--nodedir` option specifies contains only one node directory. Otherwise, this option is required. **`--force`**: Specifies whether the instance is forcibly stopped immediately.
Possible values are as follows:
`true`;; The instance is forcibly stopped immediately (default). `false`;; The

subcommand waits until all threads that are associated with the instance are exited before stopping the instance. `--kill`: Specifies whether the instance is killed by using functionality of the operating system to terminate the instance process.
Possible values are as follows:
`false`;; The instance is not killed. The subcommand uses functionality of the Java platform to terminate the instance process (default). `true`;; The instance is killed. The subcommand uses functionality of the operating system to terminate the instance process.

Operands

instance-name: The name of the instance to stop.

Examples

Example 1 Stopping an Instance Locally

This example stops the instance `yml-i-sj01` on the machine where the subcommand is run.

```
asadmin> stop-local-instance --node sj01 yml-i-sj01
Waiting for the instance to stop ...
Command stop-local-instance executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-instance,
- create-local-instance,
- delete-instance,
- delete-local-instance,
- start-instance,

- start-local-instance,
- stop-instance

Undeploy

Removes a deployed component.

Synopsis

```
asadmin [asadmin-options] undeploy [--help]
[--target target]
[--cascade={false|true}] name
```

Description

The `undeploy` subcommand uninstalls a deployed application or module and removes it from the repository.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--cascade`**: If set to `true`, deletes all the connection pools and connector resources associated with the resource adapter being undeployed. If set to `false`, the undeploy fails if any pools and resources are still associated with the resource adapter. Then, either those pools and resources must be deleted explicitly, or the option must be set to `true`. If the option is set to `false`, and if there are no pools and resources still associated with the resource adapter, the resource adapter is undeployed. This option is applicable to connectors (resource adapters) and applications. Default value is false. **`--target`**: Specifies the target from which you are undeploying. Valid values are:
`server`;; Undeploys the component from the default server instance `server` and is the default value. `domain`;; Undeploys the component from the domain. cluster_name;; Undeploys the component from every server instance in the cluster. instance_name;; Undeploys the component from a particular stand-alone server instance.

Operands

name: Name of the deployed component.

The name can include an optional version identifier, which follows the name and is separated from the name by a colon (`:`). The version identifier must begin with a letter or number. It can contain alphanumeric characters plus underscore (`_`), dash (`-`), and period (`.`) characters. To delete multiple versions, you can use an asterisk (`*`) as a wildcard character. For more information about module and application versions, see Module and Application Versions in Payara Server Application Deployment section.

Examples

Example 1 Undeploying an Enterprise Application

This example undeploys an enterprise application named `Cart.ear`.

```
asadmin> undeploy Cart
Command undeploy executed successfully.
```

Example 2 Undeploying a Connector (Resource Adapter)

This example undeploys the connector module named `jdbcra` and performs a cascading delete to remove the associated resources and connection pools.

```
asadmin> undeploy --cascade=true jdbcra
Command undeploy executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- deploy,
- redeploy

Unfreeze-Transaction-Service

Resumes all suspended transactions.

Synopsis

```
asadmin [asadmin-options] unfreeze-transaction-service [--help]
[--target target]
```

Description

The `unfreeze-transaction-service` subcommand restarts the transaction subsystem and resumes all in-flight transactions. Invoke this subcommand on an already frozen transaction subsystem.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. `--help`: `-?`: Displays the help text for the subcommand. `--target`: This option specifies the target on which you are unfreezing the transaction service. Valid values are:
`server`;; Unfreezes the transaction service for the default server instance `server` and is the default value. configuration_name;; Unfreezes the transaction service for all server instances that use the named configuration. cluster_name;; Unfreezes the transaction service for every server instance in the cluster. instance_name;; Unfreezes the transaction service for a particular server instance.

Examples

Example 1 Using unfreeze-transaction-service

```
% asadmin unfreeze-transaction-service
Command unfreeze-transaction-service executed successfully
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- freeze-transaction-service,

- recover-transactions,
- rollback-transaction
- administering-transactions in Payara Server General Administration section
- Transactions

Uninstall-Node-Ssh

Uninstalls Payara Server software from specified SSH-enabled hosts.

Synopsis

```
asadmin [asadmin-options] uninstall-node-ssh [--help]
[--installdir as-install-parent]
[--sshport ssh-port] [--sshuser ssh-user]
[--sshkeyfile ssh-keyfile]
[--force={false|true}]
host-list
```

Description

The `uninstall-node-ssh` subcommand uninstalls Payara Server software from the hosts that are specified as the operand of the subcommand. This subcommand requires secure shell (SSH) to be configured on the host where the subcommand is run and on each host where the Payara Server software is being uninstalled.

NOTE	This subcommand is equivalent to the uninstall-node subcommand.

By default, if any node except the predefined node `localhost-`domain resides on any host from which Payara Server software is being uninstalled, the subcommand fails. To uninstall the Payara Server software from a host on which user-defined nodes reside, set the `--force` option to `true`. If the `--force` option is `true`, the subcommand removes the entire content of the parent of the base installation directory.

If a file under the parent of the base installation directory is open, the subcommand fails.

If multiple hosts are specified, the configuration of the following items must be the same on all hosts:

- Parent of the base installation directory for the Payara Server software
- SSH port
- SSH user
- SSH key file

The subcommand does not modify the configuration of the domain administration server (DAS).

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--installdir`**: The absolute path to the parent of the base installation directory where the Payara Server software is installed on each host, for example, `/export/payara7/`.
The user that is running this subcommand must have write access to the specified directory. Otherwise, an error occurs.
The specified directory must contain the installation of the Payara Server software on the host. Otherwise, an error occurs.
The default is the parent of the base installation directory of the Payara Server software on the host where this subcommand is run. **`--sshport`**: The port to use for SSH connections to the host where the Payara Server software is to be uninstalled. The default is 22. **`--sshuser`**: The user on the host where the Payara Server software is to be uninstalled that is to run the process for connecting through SSH to the host. The default is the user that is running this subcommand. To ensure that the DAS can read this user's SSH private key file, specify the user that is running the DAS process.
`--sshkeyfile`: The absolute path to the SSH private key file for user that the `--sshuser` option specifies. This file is used for authentication to the `sshd` daemon on the host.
The user that is running this subcommand must be able to reach the path to the key file and read the key file.

The default is a key file in the user's `.ssh` directory. If multiple key files are found, the subcommand uses the following order of preference:
. `id_rsa` . `id_dsa` . `identity` **`--force`**: Specifies whether the subcommand uninstalls the Payara Server software from a host even if a user-defined node resides on the host. Possible values are as follows:
`false`;; If a user-defined node resides on a host, the software is not uninstalled and the subcommand fails (default).
If the `--force` option is `false`, the subcommand removes only the Payara Server software files. Other content if the parent of the base installation directory, such as configuration files, are not removed. `true`;; The subcommand uninstalls the Payara Server software from the host even if a user-defined node resides on the host.
If the `--force` option is `true`, the subcommand removes the entire content of the parent of the base installation directory.

Operands

host-list: A space-separated list of the names of the hosts from which the Payara Server software is to be uninstalled.

Examples

Example 1 Uninstalling Payara Server Software From the Default Location

This example uninstalls Payara Server software on the hosts `sj03.example.com` and `sj04.example.com` from the default location.

```
asadmin> uninstall-node-ssh sj03 sj04
Successfully connected to gfuser@sj03.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Successfully connected to gfuser@sj04.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Command uninstall-node-ssh executed successfully.
```

Example 2 Forcibly Uninstalling Payara Server Software

This example uninstalls Payara Server software on the host `sj02.example.com`.

The software is uninstalled even if a user-defined node resides on the host. The entire

content of the `/export/payara7` directory is removed.

Some lines of output are omitted from this example for readability.

```
asadmin> uninstall-node-ssh --force --installdir /export/payara7 sj02.example.com
Successfully connected to gfuser@sj02.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Force removing file /export/payara7/mq/lib/help/en/add_overrides.htm
Force removing file /export/payara7/mq/lib/help/en/add_connfact.htm
...
Force removing directory /export/payara7/glassfish/lib/appclient
Force removing directory /export/payara7/glassfish/lib
Force removing directory /export/payara7/glassfish
Command uninstall-node-ssh executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- install-node,
- install-node-ssh,
- uninstall-node

Uninstall-Node

Uninstalls Payara Server software from specified hosts.

Synopsis

```
asadmin [asadmin-options] uninstall-node [--help]
[--installdir as-install-parent]
[--sshport ssh-port] [--sshuser ssh-user]
[--sshkeyfile ssh-keyfile]
[--force={false|true}]
host-list
```

Description

The `uninstall-node` subcommand uninstalls Payara Server software from the hosts that are specified as the operand of the subcommand. This subcommand requires secure shell (SSH) to be configured on the host where the subcommand is run and on

each host where the Payara Server software is being uninstalled.

NOTE	This subcommand is equivalent to the uninstall-node-ssh subcommand.

By default, if any node except the predefined node `localhost-`domain resides on any host from which Payara Server software is being uninstalled, the subcommand fails. To uninstall the Payara Server software from a host on which user-defined nodes reside, set the `--force` option to `true`. If the `--force` option is `true`, the subcommand removes the entire content of the parent of the base installation directory.

If a file under the parent of the base installation directory is open, the subcommand fails.

If multiple hosts are specified, the configuration of the following items must be the same on all hosts:

- Parent of the base installation directory for the Payara Server software
- SSH port
- SSH user
- SSH key file

The subcommand does not modify the configuration of the domain administration server (DAS).

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--installdir`**: The absolute path to the parent of the base installation directory where the Payara Server software is installed on each host, for example, `/export/payara7/`.
The user that is running this subcommand must have write access to the specified

directory. Otherwise, an error occurs.
The specified directory must contain the installation of the Payara Server software on the host. Otherwise, an error occurs.
The default is the parent of the base installation directory of the Payara Server software on the host where this subcommand is run. **`--sshport`**: The port to use for SSH connections to the host where the Payara Server software is to be uninstalled. The default is 22. **`--sshuser`**: The user on the host where the Payara Server software is to be uninstalled that is to run the process for connecting through SSH to the host. The default is the user that is running this subcommand. To ensure that the DAS can read this user's SSH private key file, specify the user that is running the DAS process.
`--sshkeyfile`: The absolute path to the SSH private key file for user that the `--sshuser` option specifies. This file is used for authentication to the `sshd` daemon on the host.
The user that is running this subcommand must be able to reach the path to the key file and read the key file.
The default is a key file in the user's `.ssh` directory. If multiple key files are found, the subcommand uses the following order of preference:
. `id_rsa` . `id_dsa` . `identity` **`--force`**: Specifies whether the subcommand uninstalls the Payara Server software from a host even if a user-defined node resides on the host. Possible values are as follows:
`false`;; If a user-defined node resides on a host, the software is not uninstalled and the subcommand fails (default).
If the `--force` option is `false`, the subcommand removes only the Payara Server software files. Other content if the parent of the base installation directory, such as configuration files, are not removed. `true`;; The subcommand uninstalls the Payara Server software from the host even if a user-defined node resides on the host.
If the `--force` option is `true`, the subcommand removes the entire content of the parent of the base installation directory.

Operands

host-list: A space-separated list of the names of the hosts from which the Payara Server software is to be uninstalled.

Examples

Example 1 Uninstalling Payara Server Software From the Default Location

This example uninstalls Payara Server software on the hosts `sj03.example.com` and `sj04.example.com` from the default location.

```
asadmin> uninstall-node sj03 sj04
Successfully connected to gfuser@sj03.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Successfully connected to gfuser@sj04.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Command uninstall-node executed successfully.
```

Example 2 Forcibly Uninstalling Payara Server Software

This example uninstalls Payara Server software on the host `sj02.example.com`.

The software is uninstalled even if a user-defined node resides on the host. The entire content of the `/export/payara7` directory is removed.

Some lines of output are omitted from this example for readability.

```
asadmin> uninstall-node --force --installdir /export/payara7 sj02.example.com
Successfully connected to gfuser@sj02.example.com using keyfile /home/gfuser
/.ssh/id_rsa
Force removing file /export/payara7/mq/lib/help/en/add_overrides.htm
Force removing file /export/payara7/mq/lib/help/en/add_connfact.htm
...
Force removing directory /export/payara7/glassfish/lib/appclient
Force removing directory /export/payara7/glassfish/lib
Force removing directory /export/payara7/glassfish
Command uninstall-node executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- install-node,
- install-node-ssh,

- uninstall-node-ssh

Unset-Web-Context-Param

Unsets a servlet context-initialization parameter of a deployed web application or module.

Synopsis

```
asadmin [asadmin-options] unset-web-context-param [--help]
--name=context-param-name application-name[/module]
```

Description

The `unset-web-context-param` subcommand unsets a servlet context-initialization parameter of one of the following items:

- A deployed web application
- A web module in a deployed Jakarta EE application

When a parameter is unset, its value reverts to the value, if any, that is set in the application's deployment descriptor.

The application must already be deployed. Otherwise, an error occurs.

The parameter must have previously been set by using the `set-web-context-param` subcommand. Otherwise, an error occurs.

NOTE

Do not use the `unset-web-context-param` subcommand to change the value of a servlet context-initialization parameter that is set in an application's deployment descriptor. Instead, use the set-web-context-param subcommand for this purpose.

This subcommand enables you to change the configuration of a deployed application without the need to modify the application's deployment descriptors and repackage and redeploy the application.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--name`**: The name of the servlet context-initialization parameter that is to be unset. This parameter must have previously been set by using the `set-web-context-param` subcommand. Otherwise, an error occurs.

Operands

application-name: The name of the application. This name can be obtained from the Administration Console or by using the list-applications subcommand.
The application must already be deployed. Otherwise, an error occurs. **module**: The relative path to the module within the application's enterprise archive (EAR) file. The path to the module is specified in the `module` element of the application's `application.xml` file.
module is required only if the servlet context-initialization parameter applies to a web module of a Jakarta EE application. If specified, module must follow application-name, separated by a slash (/).
For example, the `application.xml` file for the `myApp` application might specify the following web module:

+

```
<module>
    <web>
        <web-uri>myWebModule.war</web-uri>
    </web>
</module>
```

```
The module would be specified as the operand of this command as
`myApp/myWebModule.war`.
```

Examples

Example 1 Unsetting a Servlet Context-Initialization Parameter for a Web Application

This example unsets the servlet context-initialization parameter `javax.faces.STATE_SAVING_METHOD` of the web application `basic-ezcomp`. The

parameter reverts to the value, if any, that is defined in the application's deployment descriptor.

```
asadmin> unset-web-context-param
--name=javax.faces.STATE_SAVING_METHOD basic-ezcomp

Command unset-web-context-param executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- list-applications,
- list-web-context-param,
- set-web-context-param

Unset-Web-Env-Entry

Unsets an environment entry for a deployed web application or module.

Synopsis

```
asadmin [asadmin-options] unset-web-env-entry [--help]
--name=env-entry-name application-name[/module]
```

Description

The `unset-web-env-entry` subcommand unsets an environment entry for one of the following items:

- A deployed web application
- A web module in a deployed Jakarta EE application

When an entry is unset, its value reverts to the value, if any, that is set in the application's deployment descriptor.

The application must already be deployed. Otherwise, an error occurs.

The entry must have previously been set by using the set-web-env-entry subcommand. Otherwise, an error occurs.

NOTE	Do not use the `unset-web-env-entry` subcommand to change the value of an environment entry that is set in an application's deployment descriptor. Instead, use the `set-web-env-entry` subcommand for this purpose.

This subcommand enables you to change the configuration of a deployed application without the need to modify the application's deployment descriptors and repackage and redeploy the application.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--name`**: The name of the environment entry that is to be unset. The name is a JNDI name relative to the `java:comp/env` context. The name must be unique within a deployment component. This entry must have previously been set by using the `set-web-env-entry` subcommand. Otherwise, an error occurs.

Operands

application-name: The name of the application. This name can be obtained from the Administration Console or by using the list-applications subcommand.
The application must already be deployed. Otherwise, an error occurs. **module**: The relative path to the module within the application's enterprise archive (EAR) file. The path to the module is specified in the `module` element of the application's `application.xml` file.
module is required only if the environment entry applies to a web module of a Java EE application. If specified, module must follow application-name, separated by a slash (/). For example, the `application.xml` file for the `myApp` application might specify the following web module:

+

```
<module>
    <web>
        <web-uri>myWebModule.war</web-uri>
    </web>
</module>
```

+ The module would be specified as the operand of this command as `myApp/myWebModule.war`.

Examples

Example 1 Unsetting an Environment Entry for a Web Application

This example unsets the environment entry `Hello User` of the web application `hello`. The entry reverts to the value, if any, that is defined in the application's deployment descriptor.

```
asadmin> unset-web-env-entry --name="Hello User" hello

Command unset-web-env-entry executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- list-applications,
- list-web-env-entry,
- set-web-env-entry

Unset

Removes one or more variables from the multimode environment.

Synopsis

```
asadmin [asadmin-options] unset [--help]
```

```
variable-list
```

Description

The `unset` subcommand removes one or more environment variables that are set for the multimode environment. After removal, the variables and their associated values no longer apply to the multimode environment.

To list the environment variables that are set, use the `export` subcommand without options. If the `export` subcommand lists no environment variables, no environment variables are set.

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

variable-list: A space-separated list of the environment variables to remove.

Examples

Example 1 Listing the Environment Variables That Are Set

This example uses the `export` subcommand to list the environment variables that have been set.

```
asadmin> export
AS_ADMIN_USER = admin
AS_ADMIN_HOST = bluestar
AS_ADMIN_PREFIX = server1.jms-service
AS_ADMIN_PORT = 8000
Command export executed successfully
```

Example 2 Removing an Environment Variable

This example removes the `AS_ADMIN_PREFIX` environment variable.

```
asadmin> unset AS_ADMIN_PREFIX
Command unset executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- export,
- multimode

Update-Connector-Security-Map

Modifies a security map for the specified connector connection pool.

Synopsis

```
asadmin [asadmin-options] update-connector-security-map [--help]
--poolname connector_connection_pool_name
[--addprincipals principal_name1[,principal_name2]*]
[--addusergroups user_group1[,user_group2]*]
[--removeprincipals principal_name1[,principal_name2]*]
[--removeusergroups user_group1[,user_group2]*]
[--mappedusername username]
mapname
```

Description

The `update-connector-security-map` subcommand modifies a security map for the specified connector connection pool.

For this subcommand to succeed, you must have first created a connector connection pool using the `create-connector-connection-pool` subcommand.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--poolname`**: Specifies the name of the connector connection pool to

which the security map that is to be updated belongs. `--addprincipals`: Specifies a comma-separated list of EIS-specific principals to be added. Use either the `--addprincipals` or `--addusergroups` options, but not both in the same command. `--addusergroups`: Specifies a comma-separated list of EIS user groups to be added. Use either the `--addprincipals` or `--addusergroups` options, but not both in the same command. `--removeprincipals`: Specifies a comma-separated list of EIS-specific principals to be removed. `--removeusergroups`: Specifies a comma-separated list of EIS user groups to be removed. `--mappedusername`: Specifies the EIS username.

Operands

mapname: The name of the security map to be updated.

Examples

Example 1 Updating a Connector Security Map

This example adds principals to the existing security map named `securityMap1`.

```
asadmin> update-connector-security-map --poolname connector-pool1
--addprincipals principal1,principal2 securityMap1
Command update-connector-security-map executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-connector-security-map,
- delete-connector-security-map,
- list-connector-security-maps

Update-Connector-Work-Security-Map

Modifies a work security map for the specified resource adapter.

Synopsis

```
asadmin [asadmin-options] update-connector-work-security-map [--help]
--raname raname
[--addprincipals eis-principal1=server-principal1[, eis-principal2=server-
principal2]*]
[--addgroups eis-group1=server-group1[, eis-group2=server-group2]*]
[--removeprincipals eis-principal1[,eis-principal2]*]
[--removegroups eis-group1[, eis-group2]*]
mapname
```

Description

The `update-connector-work-security-map` subcommand modifies a security map for the specified resource adapter.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--addgroups`**: Specifies a comma-separated list of EIS groups to be added. Use either the `--addprincipals` option or the `--addgroups` option, but not both. **`--addprincipals`**: Specifies a comma-separated list of EIS-specific principals to be added. Use either the `--addprincipals` option or the `--addgroups` option, but not both. **`--removegroups`**: Specifies a comma-separated list of EIS groups to be removed. **`--removeprincipals`**: Specifies a comma-separated list of EIS-specific principals to be removed. **`--raname`**: Indicates the connector module name with which the work security map is associated.

Operands

mapname: The name of the work security map to be updated.

Examples

Example 1 Updating a Connector Work Security Map

This example updates `workSecurityMap2` by removing `eis-group-2`.

```
asadmin> update-connector-work-security-map
--raname my-resource-adapter --removegroups eis-group-2 workSecurityMap2
```

```
Command update-connector-work-security-map executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-connector-work-security-map,
- delete-connector-work-security-map,
- list-connector-work-security-maps

Update-File-User

Updates a current file user as specified.

Synopsis

```
asadmin [asadmin-options] update-file-user [--help]
[--groups user_groups[:user_groups]*]
[--target target
[--authrealmname authrealm_name]
username
```

Description

Th `update-file-user` subcommand updates an existing entry in the keyfile using the specified user name, password and groups. Multiple groups can be entered by separating them, with a colon (:).

If a new password is not provided, this subcommand fails if secure administration is enabled and the user being updated is an administrative user.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--groups`**: This is the name of the group to which the file user belongs. **`--authrealmname`**: Name of the authentication realm where the user to be updated can be found. **`--target`**: This option helps specify the target on which you are

updating a file user. Valid values are:

`server`;; Updates the file user in the default server instance. This is the default value.

cluster_name;; Updates the file user on every server instance in the cluster.

instance_name;; Updates the file user on a specified sever instance.

Operands

username: This is the name of the file user to be updated.

Examples

Example 1 Updating a User's Information in a File Realm

The following example updates information for a file realm user named `sample_user`.

```
asadmin> update-file-user
--groups staff:manager:engineer sample_user
Command update-file-user executed successfully
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- create-file-user,
- delete-file-user,
- list-file-groups,
- list-file-users

Update-Node-Config

Updates the configuration data of a node.

Synopsis

```
asadmin [asadmin-options] update-node-config [--help]
[--nodehost node-host]
[--installdir as-install-parent] [--nodedir node-dir]
node-name
```

Description

The `update-node-config` subcommand updates the configuration data of a node.

This subcommand can update any node, regardless of whether the node is enabled for remote communication. If a node that is enabled for remote communication is updated, the node is not enabled for remote communication after the update.

Options of this subcommand specify the new values of the node's configuration data. The default for these options is to leave the existing value unchanged.

This subcommand does not require the secure shell (SSH) to be configured to update the node. You may run this subcommand from any host that can contact the DAS.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--nodehost`**: The name of the host that the node is to represent after the node is updated. **`--installdir`**: The full path to the parent of the base installation directory of the Payara Server software on the host, for example, `/export/payara7`. **`--nodedir`**: The path to the directory that is to contain Payara Server instances that are created on the node. If a relative path is specified, the path is relative to the as-install directory, where as-install is the base installation directory of the Payara Server software on the host.

Operands

node-name: The name of the node to update. The node must exist. Otherwise, an error occurs.

Examples

Example 1 Updating the Host That a Node Represents

This example updates the host that the node `sj04` represents to `hsj04`.

```
asadmin> update-node-config --nodehost hsj04 sj04
```

```
Command update-node-config executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-node-config,
- create-node-ssh,
- delete-node-config,
- delete-node-ssh,
- install-node,
- install-node-ssh,
- list-nodes,
- uninstall-node,
- uninstall-node-ssh,
- update-node-ssh

Update-Node-Docker

Updates the configuration data of a DOCKER node.

Synopsis

```
asadmin [asadmin-options] update-node-docker [--help]
[--installdir as-install-parent]
[--nodedir node-dir]
[--dockerpasswordfile filepath]
[--dockerimage imageName]
[--dockerport port]
[--useTls={false|true}]
node-name
```

Description

The `update-node-docker` subcommand updates the configuration data of a DOCKER

node. The domain administration server (DAS) contacts a Docker node's host through the Docker engine's REST API on its specific configured host and port. By default, the subcommand fails and the node is not created if the DAS cannot reach the configured Docker engine's process.

The setup of a Docker node is more involved that other types of nodes, so it is recommended reading about the requirements and conditions for this command to work properly in creating-a-docker-node in Payara Server's High Availability section.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--nodehost`**: The name of the host where the Docker engine process is listening through its REST API. This option is mandatory, otherwise the command fails. **`--installdir`**: The full path to the parent of the base installation directory of the Payara Server software on the host, for example, `/export/payara/`. The default is the parent of the default base installation directory of the Payara Server software for the DAS.

+ When creating a Docker node this parameter is mostly irrelevant when using the official Payara Server Node images. Use this parameter to point to custom server installations in case of using different images. **`--nodedir`**: The path to the directory that is to contain Payara Server instances that are created on the node. The default is `as-install/nodes`, where as-install is the base installation directory of the Payara Server software on the host.

+ If a relative path is specified, the path is relative to the `as-install` directory. **`--dockerPasswordFile`**: This is the fully-qualified path of the password file that the Docker instance will use for authentication against the DAS.

+ Please note, that this path should be the path to the file on the **remote** machine where . This file should be a standard Payara Server password file as you would use with

asadmin. This must be specified, as Docker instances require secure admin to be enabled to start. `--dockerImage`: The Docker image to use for creating the Docker container.

+ Defaults to `payara/server-node:7.2.0`. `--dockerPort`: The port where the Docker engine's REST API endpoint is exposed.

+ Defaults to `2376`. `--useTls`: Specifies whether the Docker engine's REST API endpoint is contacted through secure communication, as recommended in most environments. Defaults to `false`.

Operands

node-name: The name of the node to update. The node must exist. Otherwise, an error occurs.

Examples

Example 1 Updating the Host where the Docker Engine REST API is hosted

This example updates the host where the Docker Engine's REST API is hosted for the `docker-node-1` node:

```
asadmin update-node-docker --nodehost test.payara.fish docker-node-1
Command update-node-docker executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-node-docker,
- delete-node-docker,
- list-nodes

Update-Node-Ssh

Updates the configuration data of a SSH node.

Synopsis

```
asadmin [asadmin-options] update-node-ssh [--help]
[--nodehost node-host]
[--installdir as-install-parent] [--nodedir node-dir]
[--sshport ssh-port] [--sshuser ssh-user]
[--sshkeyfile ssh-keyfile]
[--force={false|true}]
node-name
```

Description

The `update-node-ssh` subcommand updates the configuration data of an SSH node. This subcommand requires secure shell (SSH) to be configured on the machine where the domain administration server (DAS) is running and on the machine where the node resides. You may run this subcommand from any machine that can contact the DAS.

This subcommand can update any node, regardless of whether the node is enabled for remote communication. If the node is not enabled for remote communication, the subcommand enables SSH communication for the node and updates any other specified configuration data.

Options of this subcommand specify the new values of the node's configuration data. The default for most options is to leave the existing value unchanged. However, if this subcommand is run to enable SSH communication for a node, default values are applied if any of the following options is omitted:

- `--sshport`
- `--sshuser`
- `--sshkeyfile`

By default, the subcommand fails and the node is not updated if the DAS cannot contact the node's host through SSH. To force the node to be updated even if the host cannot be contacted through SSH, set the `--force` option to `true`.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--nodehost`**: The name of the host that the node is to represent after the node is updated. **`--installdir`**: The full path to the parent of the base installation directory of the Payara Server software on the host, for example, `/export/payara`. **`--nodedir`**: The path to the directory that is to contain Payara Server instances that are created on the node. If a relative path is specified, the path is relative to the as-install directory, where as-install is the base installation directory of the Payara Server software on the host. **`--sshport`**: The port to use for SSH connections to this node's host. The default depends on whether this subcommand is run to enable SSH communication for the node:
* If the node is already enabled for communication over SSH, the default is to leave the port unchanged. * If this subcommand is run to enable SSH communication for the node, the default port is `22`.
If the `--nodehost` is set to `localhost`, the `--sshport` option is ignored.
`--sshuser`: The user on this node's host that is to run the process for connecting to the host through SSH. The default depends on whether this subcommand is run to enable SSH communication for the node:
* If the node is already enabled for communication over SSH, the default is to leave the user unchanged. * If this subcommand is run to enable SSH communication for the node, the default is the user that is running the DAS process.
If the `--nodehost` option is set to `localhost`, the `--sshuser` option is ignored.
`--sshkeyfile`: The absolute path to the SSH private key file for user that the `--sshuser` option specifies. This file is used for authentication to the `sshd` daemon on the node's host.

NOTE	Payara Server also supports password authentication through the `AS_ADMIN_SSHPASSWORD` entry in the password file. The password file is specified in the `--paswordfile` option of the asadmin-1m utility.

If the SSH private key file is protected by a passphrase, the password file must contain the `AS_ADMIN_SSHKEYPASSPHRASE` entry. The path to the key file must be reachable by the DAS and the key file must be readable by the DAS.
The default depends on whether this subcommand is run to enable SSH communication for the node:
* If the node is already enabled for communication over SSH, the default is to leave the key file unchanged. * If this subcommand is run to enable SSH communication for the node, the default is the key file in the user's `.ssh` directory. If multiple key files are found, the subcommand uses the following order of preference: . `id_rsa` . `id_dsa` . `identity` **`--force`**: Specifies whether the node is updated even if validation of the node's parameters fails. To validate a node's parameters, the DAS must be able to contact the node's host through SSH. Possible values are as follows:
`false`;; The node is not updated if validation of the node's parameters fails (default).
`true`;; The node is updated even if validation of the node's parameters fails.

Operands

node-name: The name of the node to update. The node must exist. Otherwise, an error occurs.

Examples

Example 1 Updating the Host That a Node Represents

This example updates the host that the node `lssh` represents to `sj04`.

```
asadmin> update-node-ssh --nodehost sj04 lssh
Command update-node-ssh executed successfully.
```

Example 2 Forcing the Update of a Node

This example forces the update of the node `sj01` to enable the node to communicate over SSH.

```
asadmin> update-node-ssh --force sj01
Warning: some parameters appear to be invalid.
Could not connect to host sj01 using SSH.
Could not authenticate. Tried authenticating with specified key at
```

```
/home/gfuser/.ssh/id_rsa
Continuing with node update due to use of --force.
Command update-node-ssh executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-node-config,
- create-node-ssh,
- delete-node-config,
- delete-node-ssh,
- install-node,
- install-node-ssh,
- list-nodes,
- uninstall-node,
- uninstall-node-ssh,
- update-node-config

Update-Password-Alias

Updates a password alias.

Synopsis

```
asadmin [asadmin-options] update-password-alias [--help]
aliasname
```

Description

This subcommand updates the password alias IDs in the named target. An alias is a token of the form `${ALIAS=password-alias-password}`. The password corresponding to the alias name is stored in an encrypted form.The `update-`

`password-alias` subcommand takes both a secure interactive form (in which the user is prompted for all information) and a more script-friendly form, in which the password is propagated on the command line.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Operands

aliasname: This is the name of the password as it appears in `domain.xml`.

Examples

Example 1 Updating a Password Alias

```
asadmin> update-password-alias jmspassword-alias
Please enter the alias password>
Please enter the alias password again>
Command update-password-alias executed successfully.
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- create-password-alias,
- delete-password-alias,
- list-password-aliases

Uptime

Returns the length of time that the DAS has been running.

Synopsis

```
asadmin [asadmin-options] uptime [--help]
```

Description

The `uptime` subcommand returns the length of time that the domain administration server (DAS) has been running since it was last restarted.

This subcommand is supported in remote mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand.

Examples

Example 1 Showing How Long the DAS Has Been Running

This example shows the length of time that the DAS has been running.

```
asadmin> uptime
Uptime: 2 days, 1 hours, 30 minutes, 18 seconds, Total milliseconds: 178218706
Command uptime executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-domains,
- start-domain,
- stop-domain

Validate-Multicast

Validates that multicast transport is available for clusters.

Synopsis

```
asadmin [asadmin-options] validate-multicast [--help]
[--multicastport multicastport]
[--multicastaddress multicastaddress]
[--bindaddress bindaddress]
[--sendperiod sendperiod]
[--timeout timeout]
[--timetolive timetolive]
[--verbose={false|true}]
```

Description

The `validate-multicast` subcommand validates that multicast transport is available for clusters. You should run this subcommand at the same time on each of the hosts to be validated. This subcommand is available in local mode.

NOTE

Do not run the `validate-multicast` subcommand using the DAS and cluster's multicast address and port values while the DAS and cluster are running. Doing so results in an error.
The `validate-multicast` subcommand must be run at the same time on two or more machines to validate whether multicast messages are being received between the machines.

As long as all machines see each other, multicast is validated to be working properly across the machines. If the machines are not seeing each other, set the `--bindaddress` option explicitly to ensure that all machines are using interface on same subnet, or increase the `--timetolive` option from the default of `4`. If these changes fail to resolve the multicast issues, ask the network administrator to verify that the network is configured so the multicast messages can be seen between all the machines used to run the cluster.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--multicastport`**: The port for the multicast socket on which the Group Management Service (GMS) listens for group events. Specify a standard UDP port number in the range 2048-49151. The default is `2048`. **`--multicastaddress`**:

The address for the multicast socket on which the GMS listens for group events. Specify a class D IP address. Class D IP addresses are in the range `224.0.0.0` to `239.255.255.255`, inclusive. The address `224.0.0.0` is reserved and should not be used. The default is `228.9.3.1`. **`--bindaddress`**: The local interface to receive multicast datagram packets for the GMS. The default is to use all available binding interfaces.

On a multi-home machine (possessing two or more network interfaces), this attribute enables you to indicate which network interface is used for the GMS. This value must be a local network interface IP address. **`--sendperiod`**: The number of milliseconds between test messages sent between nodes. The default is `2000`. **`--timeout`**: The number of seconds before the subcommand times out and exits. The default is `20`. You can also exit this subcommand using Ctrl-C. **`--timetolive`**: The default time-to-live for multicast packets sent out on the multicast socket in order to control the scope of the multicasts. The time-to-live value must be between zero and 255 inclusive. The default is the JDK default or a minimum defined by a constant in the GMS subsystem, whichever is lower. To see the time-to-live value being used, use the `--verbose` option. **`--verbose`**: If used without a value or set to `true`, provides additional debugging information. The default is `false`.

Examples

Example 1 Validating multicast transport

Run from host `machine1`:

```
asadmin> validate-multicast
Will use port 2,048
Will use address 228.9.3.1
Will use bind address null
Will use wait period 2,000 (in milliseconds)

Listening for data...
Sending message with content "machine1" every 2,000 milliseconds
Received data from machine1 (loopback)
Received data from machine2
Exiting after 20 seconds. To change this timeout, use the --timeout command line
option.
Command validate-multicast executed successfully
```

Run from host `machine2`:

```
asadmin> validate-multicast
Will use port 2,048
Will use address 228.9.3.1
Will use bind address null
Will use wait period 2,000 (in milliseconds)

Listening for data...
Sending message with content "machine2" every 2,000 milliseconds
Received data from machine2 (loopback)
Received data from machine1
Exiting after 20 seconds. To change this timeout, use the --timeout command line
option.
Command validate-multicast executed successfully
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m,
- get-health

Verify-Domain-Xml

Verifies the contents of the domain.xml file.

Synopsis

```
asadmin [asadmin-options] verify-domain-xml [--help]
[--domaindir domain-dir] [domain-name]
```

Description

Verifies the content of the `domain.xml` file by checking the following details:

- That the `domain.xml` file can be parsed
- That the names for elements that have them are unique

This subcommand is supported in local mode only.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options,

see the asadmin-1m help page. `-h --help`: Displays the help text for the subcommand. `--domaindir`: Specifies the domain root directory, where the domains are located. The path must be accessible in the file system. The default is `as-install/domains`.

Operands

domain_name: Specifies the name of the domain. The default is `domain1`.

Examples

Example 1 Using verify-domain-xml

```
asadmin> verify-domain-xml
All Tests Passed.
domain.xml is valid
```

Exit Status

0: command executed successfully **1**: error in executing the command

See Also

- asadmin-1m

Version

Displays version information for Payara Server.

Synopsis

```
asadmin [asadmin-options] version [--help]
[--verbose={false|true}]
[--local={false|true}]
```

Description

The `version` subcommand displays version information for Payara Server. By default, if the subcommand cannot contact the domain administration server (DAS), the subcommand retrieves the version information locally and displays a warning message.

This subcommand is supported in remote mode and local mode.

Options

asadmin-options: Options for the `asadmin` utility. For information about these options, see the asadmin-1m help page. **`--help`**: **`-?`**: Displays the help text for the subcommand. **`--verbose`**: **`-v`**: If this option is set to `true`, the subcommand provides the version of the Java Runtime Environment (JRE) that the server is running. The default is `false`. **`--local`**: If this option is set to `true`, the subcommand obtains the version locally from the installation of Payara Server on the host where the subcommand is run.

+ If this option is set to `false` (default), the subcommand attempts to contact the DAS to obtain the version. If the attempt to contact the DAS fails, the subcommand retrieves the version locally and displays a warning message.

Examples

Example 1 Obtaining Version Information From a Running DAS

```
asadmin> version
Version = Payara Server 7.2.0 #badassfish (build 00)
Command version executed successfully.
```

Example 2 Obtaining Version Information When the DAS Cannot be Reached

```
asadmin> version
Version string could not be obtained from Server [localhost:4848].
(Turn debugging on e.g. by setting AS_DEBUG=true in your environment, to see the
details.)
Using locally retrieved version string from version class.

Version = Payara Server 7.2.0 #badassfish (build 00)
Command version executed successfully.
```

Exit Status

0: subcommand executed successfully **1**: error in executing the subcommand

See Also

- asadmin-1m,
- list-modules

Extensions

Server Extensions

Payara Server Enterprise Edition allows plugin extensions to expand the features of the server's runtime. These extensions are not included in the default server's runtime and they must be installed before considering their usage.

Currently, the following extensions are provided:

"gRPC Support"
The gRPC Support module allows web applications to define gRPC services which can be consumed by client applications.

gRPC Support

Grpc

WARNING	The gRPC extension has not yet been updated to support Azul Payara Enterprise 6

What Is Grpc?

gRPC is a modern open-source high performance Remote Procedure Call (RPC) framework that use HTTP2 and simplifies the communication with protocol buffers to share data between client and server. By enabling gRPC Payara Server gives more options to create binary adaptable services.

What Can We Do with It?

- We can create services more efficient than REST.
- Use HTTP/2 to provide full duplex communication between the client and the service.
- Use protocol buffers to optimize and simplify communication.
- Enables bidirectional streaming out-of-the-box.

gRPC supports the means for users to implement gRPC style service programming.

See Also

- You can find detailed information about gRPC here: https://grpc.io/docs/what-is-grpc/introduction/
- You can find detailed information about to define a grpc service and client here: https://grpc.io/docs/languages/java/basics/ .
- You can find detailed information about `protobuf` protocol here: https://grpc.io/docs/what-is-grpc/introduction/#working-with-protocol-buffers.

Installing Grpc Server Support Module

WARNING	The gRPC extension has not yet been updated to support Azul Payara Enterprise 6

You need to download the following artifact from enterprise nexus repository: https://nexus.payara.fish/repository/payara-enterprise .

IMPORTANT	You must have an active Payara Enterprise subscription to access this Nexus repository.

This is the reference of the artifact to download:

```
<dependency>
  <groupId>fish.payara.extensions.grpc</groupId>
  <artifactId>grpc-support</artifactId>
  <version>1.0.0</version>
</dependency>
```

To download the artifact use the following command:

```
mvn dependency:get -DrepoUrl=https://nexus.payara.fish/repository/payara-enterprise
-Dartifact=fish.payara.extensions.grpc:grpc-support:1.0.0 -Dpackaging=jar
-Dclasifier=sources -Ddest=.
```

The previous command will download the JAR file to the current folder, then you need to drop the JAR into the `${PAYARA_HOME}/glassfish/modules` directory of your Payara Server Installation.

```
cp grpc-support-1.0.0.jar ${PAYARA_HOME}/glassfish/modules
```

Restart the domain and any running instances to enable the module.

Using Grpc Support Module

WARNING	The gRPC extension has not yet been updated to support Azul Payara Enterprise 6

This page documents how to use gRPC support in web applications.

Preparing a Grpc Service in a Web Application.

To prepare the service gRPC application you need the following:

1. Create a sample web application.
2. Include configuration for empty context root.
3. Create the `protobuf` service definition file.
4. Include build dependencies.
5. Configure build section.
6. Build service.
7. Extend service implementation.
8. Rebuild service.
9. Deploy service application.

Create a Sample Web Application

Create a sample web application project using the following Maven archetype:

Example:

```
mvn archetype:generate -DgroupId=fish.payara.samples -DartifactId=payara-web
-DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

Include Configuration for Empty Context Root

You need to create a Maven web application with empty context-root `/`. You can add the following `payara-web.xml` deployment descriptor to the `WEB-INF` folder of your

application:

```
<!DOCTYPE payara-web-app PUBLIC "-//Payara.fish//DTD Payara Server 7 Servlet 6.1
Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-web-app_6_1-1.dtd">
<payara-web-app>
 <context-root>/</context-root>
</payara-web-app>
```

IMPORTANT	gRPC services must be deployed under the context-root `/`, since this is a constraint of the gRPC style programming as standard clients expect no base path on HTTP requests.

Create the `protobuf` Service Definition File

You need to use the `protobuf` protocol in order to define your service. Here's a sample "Hello World" snippet:

```
syntax = "proto3";
option java_multiple_files = true;
option java_package = "fish.payara.samples.grpc";
option java_outer_classname = "PayaraProto";
package fish.payara.samples.grpc;
service PayaraService {
  rpc communicate (PayaraReq) returns (PayaraResp) {}
}
message PayaraReq {
  string message = 1;
}
message PayaraResp {
  string message = 1;
}
```

To summarize the snippet above, this file is defining a service named `PayaraService` with a single method named `communicate` that handles a request named `PayarReq` and a returns a response named `PayaraResp`. We can see also the structure of these objects as they each have a string field named `message`. That will be the content of the messages to process by the service.

The `protobuf` file should be added under the following folder:

- src

- main
  - proto
    - payara.proto

NOTE The name of the file is irrelevant, it only matters that its extension is `.proto`.

Include Build Dependencies

It is needed to declare the following dependencies to build the service and generate its stubs:

Example:

```
<dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-protobuf</artifactId>
    <version>1.43.1</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-stub</artifactId>
    <version>1.43.1</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>jakarta.platform</groupId>
    <artifactId>jakarta.jakartaee-api</artifactId>
    <version>10.0.0</version>
    <scope>provided</scope>
</dependency>
```

- The `grpc-protobuf` artifact is used to parse the `protobuf` files and generate the correspondent stubs class files.
- The `grpc-stub` artifact is used to resolve internal types for the stub classes.

Configure the Build Section

The build section is needed to generate the stubs and service implementation.

Example:

```
<build>
    <extensions>
        <extension>
            <groupId>kr.motd.maven</groupId>
            <artifactId>os-maven-plugin</artifactId>
            <version>1.6.2</version>
        </extension>
    </extensions>
    <plugins>
        <plugin>
            <groupId>org.xolstice.maven.plugins</groupId>
            <artifactId>protobuf-maven-plugin</artifactId>
            <version>0.6.1</version>
            <configuration>

<protocArtifact>com.google.protobuf:protoc:3.19.2:exe:${os.detected.classifier}</pr
otocArtifact>
                <pluginId>grpc-java</pluginId>
                <pluginArtifact>io.grpc:protoc-gen-grpc-
java:1.43.1:exe:${os.detected.classifier}</pluginArtifact>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                        <goal>compile-custom</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

- The `os-maven-plugin` extension is used to identify OS properties used during generation of stubs classes.
- The `protobuf-maven-plugin` plugin calls the `protobuf` compiler (`protoc`) to generate the stubs classes.

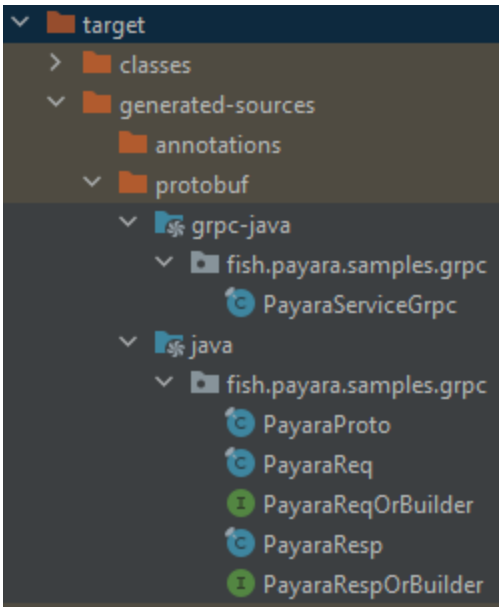
Build the Service

After doing the previous steps, you can build the project to generate the stubs. These stubs are needed to resolve the types used in the service implementation.

Example:

```
${webapp root folder} > mvn clean install
```

The stubs should be generated with similar structure like the following:



target/generated-sources/protobuf/grpc-java/fish.payara.samples.grpc: Here you can see the service class **target/generated-sources/protobuf/java/fish.payara.samples.grpc**: Here you can see the types associated to the service

Extending Service Implementation Class

To implement the service's endpoint you'll have to extend the implementation base class that is nested inside the recently compiled gRPC stub class located under `/target/generated-sources/protobuf/grpc-java`:

```
package fish.payara.samples.grpc;
import jakarta.enterprise.context.Dependent;
import java.util.logging.Logger;

@Dependent
public class PayaraService extends PayaraServiceGrpc.PayaraServiceImplBase {
    private final static Logger log =
Logger.getLogger(PayaraService.class.getName());
    @Override
    public void communicate(fish.payara.samples.grpc.PayaraReq request,

io.grpc.stub.StreamObserver<fish.payara.samples.grpc.PayaraResp> responseObserver)
{
        final String message = request.getMessage(); //getting message from the
request
        log.info(String.format("Processing message: %s", message)); //printing
incoming message from the request
        responseObserver.onNext(response(message)); //setting the message to the
response
        responseObserver.onCompleted(); //indicating that the response is complete
    }
```

```
    private static final fish.payara.samples.grpc.PayaraResp response(String
message) {
        return fish.payara.samples.grpc.PayaraResp.newBuilder() //creating builder
                .setMessage(message) //setting response message
                .build(); //build the response
    }
}
```

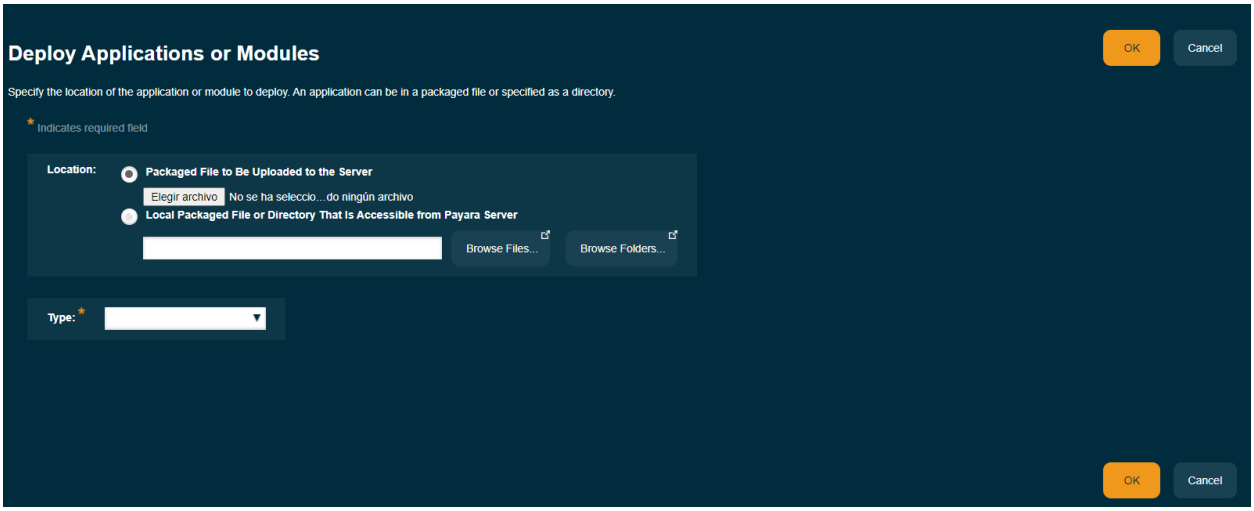
Our example is a simple "echo" service that will print a "Hello World" message. We can see that the `communicate` method receives the `PayaraReq` and the `StreamObserver<fish.payara.samples.grpc.PayaraResp>` parameters which are needed to process the incoming request with the 'Hello World' message and create the response using the same message.

Rebuild Service

After doing the previous steps, you need to rebuild the project to include the service implementation.

Deploy Service Application

The final step is to deploy the application in a Payara Server domain. We can do this by using the Admin Console or the Asadmin CLI:



Here's an example running the Asadmin CLI equivalent from the command line:

```
${PAYARA_HOME}/glassfish/bin > asadmin deploy [filelocation]/service.war
```

Creating a Sample Grpc Client Application

After deploying the service we'll proceed to create a client. To achieve this let's follow these steps:

1. Create client application
2. Copy stubs files to the client source directories
3. Add client dependencies
4. Create client implementation
5. Execution of the `HelloWorld` application

Create the Client Application

We can create a client application using the following Maven archetype:

```
mvn archetype:generate -DgroupId=fish.payara.samples -DartifactId=payara-client
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Copy Stubs Files to the Client Source Folder

Now, we'll manually copy the service stub files that were generated in the previous section to the following locations in the client application's source directories:

```
cp ${server app folder}/target/generated-sources/protobuf/grpc-
java/fish.payara.samples.grpc  ${client app}/src/main/java/fish/payara/samples/grpc

cp ${server app folder}/target/generated-
sources/protobuf/java/fish.payara.samples.grpc ${client app
}/src/main/java/fish/payara/samples/grpc
```

Add Client Dependencies

The following Maven dependencies are needed to build and run the client application:

Example:

```
<dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-netty-shaded</artifactId>
    <version>1.43.1</version>
    <scope>runtime</scope>
</dependency>
<dependency>
```

```
    <groupId>io.grpc</groupId>
    <artifactId>grpc-protobuf</artifactId>
    <version>1.43.1</version>
</dependency>
<dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-stub</artifactId>
    <version>1.43.1</version>
</dependency>
```

- The `grpc-netty-shaded` it is needed at runtime to create the communication channel and send the message to the service.
- The `grpc-protobuf` artifact is used to parse the `protobuf` files and generate the correspondent stubs class files.
- The `grpc-stub` artifact is used to resolve internal types for the stub classes.

Create Client Implementation

Finally, here is the client code to call the gRPC service deployed in Payara Server:

```
package fish.payara.samples.grpc;

import io.grpc.Channel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.stub.StreamObserver;
import java.net.MalformedURLException;
import java.net.URISyntaxException;
import java.net.URL;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicReference;
import java.util.logging.Level;
import java.util.logging.Logger;

public class GrpcClient {

    private static final Logger LOGGER =
Logger.getLogger(GrpcClient.class.getName());
    private final PayaraServiceGrpc.PayaraServiceStub stub; //reference to the stub
service implementation class
    private CountDownLatch latch;
    private AtomicReference<Throwable> error;

    public static void main(String[] args) throws InterruptedException,
MalformedURLException, URISyntaxException {
        URL myURL = new
URL("http://localhost:8080/fish.payara.samples.grpc.PayaraService"); // URL for the
deployed gRPC service
        final GrpcClient client = new GrpcClient(myURL); // creating client
        client.communicate(); // call service
    }
```

```
    public GrpcClient(URL url) throws URISyntaxException {
        final Channel channel = ManagedChannelBuilder.forAddress(url.getHost(),
url.getPort())
                .usePlaintext().build(); //creating channel to start communication
to the service
        this.stub = PayaraServiceGrpc.newStub(channel); //creating stub from the
channel reference
        this.error = new AtomicReference<>(null);
    }

    public void communicate() throws InterruptedException {
        latch = new CountDownLatch(1); //this is to wait until the communication
finish with the current thread
        stub.communicate(request("Hello World"), new ResponseObserver()); //calling
service and adding a ResponseObserver to process response
        latch.await(20, TimeUnit.SECONDS); //timeout to wait response
    }

    public Throwable getError() {
        return error.get();
    }

    private final class ResponseObserver implements StreamObserver<PayaraResp> {

        @Override
        public void onNext(PayaraResp response) { //to process the service response
            LOGGER.log(Level.INFO, "Response received: \"{0}\".",
response.getMessage()); // printing the response from the service
        }

        @Override
        public void onError(Throwable t) { //method to process errors
            LOGGER.log(Level.SEVERE, "Error received", t);
            error.set(t);
            latch.countDown();
        }

        @Override
        public void onCompleted() {
            latch.countDown(); //indicating that the communication complete for the
current thread
        }

    }

    private static final PayaraReq request(String message) {
        return PayaraReq.newBuilder().setMessage(message).build(); //creating
request with a String message
    }
}
```

Execution of the "Helloworld" Application

To execute the client application, build the project and run the following command:

```
${client app} mvn compile exec:java -Dexec.mainClass
="fish.payara.samples.grpc.GrpcClient"
```

And you'll see the following log entries on the server that the service prints after processing the corresponding message:

```
[#|2022-03-02T14:15:10.947-0600|INFO|Payara
6.2022.1|jakarta.enterprise.system.container.web.com.sun.web.security|_ThreadID=118
;_ThreadName=http-thread-pool::http-listener-
1(2);_TimeMillis=1646252110947;_LevelValue=800;|
  Context path from ServletContext:  differs from path from bundle: /|#]

[#|2022-03-02T14:15:10.990-0600|INFO|Payara
6.2022.1|fish.payara.samples.grpc.PayaraService|_ThreadID=234;_ThreadName=grpc-
default-executor-0;_TimeMillis=1646252110990;_LevelValue=800;|
  Processing message: Hello World|#]
```

While the following entries are printed out on the client side:

```
mar 02, 2022 2:26:37 PM fish.payara.samples.grpc.GrpcClient$ResponseObserver onNext
INFO: Response received: "Hello World".
```

This shows that both client and server are running correctly.

Payara Server Docker Image Overview

Overview of the usage and details of the official Payara Server Enterprise Docker image.

Quick Start

Launch a new Payara Server domain running in a container listening in port `8080` with the following command:

```
docker run -p 8080:8080 nexus.payara.fish:5000/payara/server-full
```

Keep in mind the following:

- The container starts the `production` domain in the foreground so the server's DAS becomes the main process.
- The *tini* utility is used to guarantee that the server runs seamlessly as the single child process.
- The server's JVM process runs under the `payara` user.

IMPORTANT	It is recommended to specify the tag of the correct Payara Enterprise version that will be used on your production environment for guaranteed long-term stability

Using a specific version

```
docker run -p 8080:8080 nexus.payara.fish:5000/payara/server-full:7.2.0
```

AMD64 & ARM64 compatible images are provided. The Docker client should automatically download the appropriate image for your architecture.

Other Distributions

Besides the Full Profile version, the Payara Server Web Profile distribution is also available as a Docker Image under the `payara/server-web` name. The image for this distribution behaves the same as its Full Profile counterpart.

Exposed Ports

The default ports that are exposed by this image are:

- 8080 - HTTP listener
- 8181 - HTTPS listener
- 4848 - Admin Service HTTPS listener
- 9009 - JDWP Debug port

Administration Details

To access the DAS' administration interface (either by remote asadmin commands or by using the Web Admin Console), just map the container's Admin Service port when launching the container:

```
docker run -p 4848:4848 -p 8080:8080 nexus.payara.fish:5000/payara/server-
full:7.2.0
```

By default, the admin service will be secure-enabled, meaning that HTTPS communication will be used exclusively.

IMPORTANT	The default administrator user with access to the admin interface will use the `admin/admin` credentials.
CAUTION	Make sure to update the administrator credentials after the domain has started.

Installation Details

The Payara Server installation is located in the `/opt/payara/appserver` directory (identified by the `PAYARA_DIR` environment variable). The `/opt/payara/` parent directory is the default working directory of the docker image.

NOTE	The directory name is deliberately free of any versioning so that any scripts written to work with one version can be seamlessly migrated to the latest docker image.

Application Deployment

Deploy Applications from a Directory

The default entrypoint will scan the `$DEPLOY_DIR` directory (which defaults to `/opt/payara/deployments`) for files and sub-folders and deploy them automatically after the domain is started.

NOTE	Any RAR files artefacts found in the directory will always be deployed first

To deploy your applications, you can mount this directory as a Docker volume mapped to a local directory that hosts your application artefacts.

The following sample command will launch a new Payara Server container and deploy all applications located in the `~/payara/apps` directory:

```
docker run -p 8080:8080 -v ~/payara/apps:/opt/payara/deployments
nexus.payara.fish:5000/payara/server-full:7.2.0
```

Deploy Applications Using a Custom Image

You can also prepare a custom image based on the official Payara Server Docker image and copy all application artifacts to the `$DEPLOY_DIR` directory. This way, you can re-package your applications in a simple distributable unit.

The following sample *Dockerfile* can be used to prepare a custom Payara Server image that deploys the `myapplication.war` artefact at runtime:

```
FROM nexus.payara.fish:5000/payara/server-full:7.2.0
COPY myapplication.war $DEPLOY_DIR
```

And to launch the container, simply build and run the image like this:

```
docker build -t mycompany/myapplication:1.0 .
docker run -p 8080:8080 mycompany/myapplication:1.0
```

Configuration

Using Environment Variables

The following environment variables can be used to configure multiple settings of the Payara Server's DAS before the domain is started. They can be either specified in a custom image's `Dockerfile`, passed to the `docker run` command via the `--env` or `--env-file` arguments or modified by an init script before the `startInForeground.sh` script is executed:

Name	Description	Default Value
`PREBOOT_COMMANDS`	The name of the file containing commands to be executed before the domain is started. The file will not be modified by the Docker container	`$CONFIG_DIR/post-boot-commands.asadmin` and `$CONFIG_DIR/pre-boot-commands.asadmin`

Name	Description	Default Value
`POSTBOOT_COMMANDS`	The name of the file containing commands to be executed after the domain is started. This is the file written to in the `generate_deploy_commands.sh` script. The file will not be modified by the Docker container	`$CONFIG_DIR/post-boot-commands.asadmin`
`MEM_MAX_RAM_PERCENTAGE`	Value for the JVM argument `-XX:MaxRAMPercentage` which indicates the percentage of memory assigned to the container that can be used by the Java process	`70`
`MEM_XSS`	Value for the JVM argument `-Xss` which controls the stack size	`512K`
`DEPLOY_PROPS`	Specifies a list of properties to be passed with the deploy commands generated in the `generate_deploy_commands.sh` script	
`PAYARA_ARGS`	Additional arguments passed to the `start-domain` command that starts the DAS. *Use with caution.*	
`JVM_ARGS`	Additional JVM arguments which will be used to configure the Payara Servers DAS JVM settings	

IMPORTANT | Preboot and post-boot command files will not be modified by the Docker container

The following is a list of variables used by the Docker image to configure the Payara Server domain, so it is not recommended to alter their values:

Name	Description	Value
`HOME_DIR`	The home directory for the `payara` user	`/opt/payara`
`PAYARA_DIR`	The root directory of the Payara installation	`/opt/payara/appserver`

Name	Description	Value
`SCRIPT_DIR`	The directory where the `generate_deploy_commands.sh` and `startInForeground.sh` scripts can be found	`/opt/payara/scripts`
`CONFIG_DIR`	The directory where the post and pre boot files are generated to by default	`/opt/payara/config`
`DEPLOY_DIR`	The directory where applications are searched for in `generate_deploy_commands.sh` script	`/opt/payara/deployments`
`ADMIN_USER`	The default username credential for the default administrator user	`admin`
`ADMIN_PASSWORD`	The default password credential for the default administrator user. Can only be set when the basic image is created	`admin`
`PASSWORD_FILE`	The location of the password file for asadmin. This can be passed to asadmin using the `--passwordfile` parameter	`/opt/payara/passwordFile`
`DOMAIN_NAME`	The name of the domain running within the container	`production`
`AS_ADMIN_MASTERPASSWORD`	The master password to pass to Payara Server. This is overridden if one is specified in the `$PASSWORD_FILE`	`changeit`

Executing Administration Commands at Domain Startup

It's possible to run a set of custom administration commands during the domain startup. You can either specify the `PREBOOT_COMMANDS` or `POSTBOOT_COMMANDS` environment variables to point to the absolute path of a custom script file, or you can just copy the file to the expected paths (See above).

For example, the following sample command will execute all commands listed in the `post-boot-commands.asadmin` file inside the `/local/path/with/boot/file` directory mounted volume:

```
docker run -p 8080:8080 -v /local/path/with/boot/file:/config -e POSTBOOT_COMMANDS
```

```
=/config/post-boot-commands.asadmin nexus.payara.fish:5000/payara/server-full:7.2.0
```

Alternatively, the same outcome can be achieved by defining a custom Docker image:

```
FROM nexus.payara.fish:5000/payara/server-full:7.2.0
COPY post-boot-commands.asadmin $POSTBOOT_COMMANDS
```

Enable Logging to a File

By default, Payara Server logs are written to the Docker standard output instead of the usual server.log file. However, you can configure the server to also log to this server.log file using the following command with a pre-boot command file:

```
set-log-attributes com.sun.enterprise.server.logging.GFFileHandler.logtoFile=true
```

Alternatively, you can redirect the container logs to a file when running the container:

```
docker run -p 8080:8080 nexus.payara.fish:5000/payara/server-full:{release-id} >
payara-logs.txt 2>&1
```

Executing Custom Scripts before Domain Startup

If preboot command files are not enough, you can add your customized shell scripts to the `${SCRIPT_DIR}/init.d` directory. This will allow you to modify the environment before the Payara Server domain starts.

Scripts added to this folder should be Bash scripts and they should end with the `.sh` suffix.

TIP

Scripts in `${SCRIPT_DIR}/init.d` will be executed in the standard file order. If you need to start them in a specific order, we recommend prepending a prefix number to their name, for example, `01_script1.sh`, `02_script2.sh`, and so on.

As with the preboot command file, you may either create a custom Docker image that already contains the script files in question or you can also mount a volume mapped to the `${SCRIPT_DIR}/init.d` directory instead.

The Default Docker Entrypoint

The default entry point of the Docker image is defined using the tini utility, to allow the DAS's JVM to run as a single child process.

The default `CMD` argument for *tini* runs the `bin/entrypoint.sh` shell script in `exec` mode, which in turn runs the following scripts in order:

1. `${SCRIPT_DIR}/init_1_generate_deploy_commands.sh`: This script outputs deploy commands to the post-boot command file located at `$POSTBOOT_COMMANDS` (default `$CONFIG_DIR/post-boot-commands.asadmin`). If deploy commands are already found in that file, this script does nothing.
2. `${SCRIPT_DIR}/init.d/*.sh`: As described above, these scripts can be provided by you to run and configure the environment **before** the domain startup.
3. `${SCRIPT_DIR}/startInForeground.sh`. This script starts the domain in the foreground, in a manner that allows the JVM to be controlled by the docker host.

Browsing the Container

You can override the default entrypoint if needed to test or browse the container to examine that everything's in place. The following command will start the container at a bash prompt, without starting the domain.

```
docker run -p 8080:8080 -it nexus.payara.fish:5000/payara/server-full:7.2.0 bash
```

Read Only Support Considerations

The Payara Server Docker image requires write access to the `PAYARA_DIR`, `CONFIG_DIR`, and `/tmp` directories to function. If you wish to use `--read-only` Docker containers (or the equivalent `--readOnlyRootFilesystem` property in Kubernetes), you must mount volumes at these directories which the `payara` user has write access to. This can be done using anonymous or tmpfs volumes if you do not wish to persist the data outside the life of the container. Named volumes or bind mounts may be used if you wish the data to persist outside the life of the container.

Here is an example `run` command for a read-only Docker container which will delete

itself and its anonymous volumes when stopped:

```
docker run -p 8080:8080 -p 4848:4848 --read-only --rm -v /opt/payara -v /tmp
nexus.payara.fish:5000/payara/server-full:7.2.0
```

Payara Insight

Payara InSight allows real-time visualization and alerting for server and web application metrics.

It consists of a server module responsible for collecting a multitude of metrics from all nodes which are accumulated on the DAS to feed data to its frontend which visualizes the data as a web-application deployed on the DAS.

The metrics include server metrics from different sources within the server as well as metrics defined in user applications using *MicroProfile Metrics* and metrics derived from *MicroProfile Health* checks.

Enabling the GUI

By default, the Payara Insight GUI is automatically enabled for the `production` domain. Links can be found on the *Common Tasks* home page of the Admin GUI in section *Monitoring* or on the *Monitoring Data* page.

For `domain1` the GUI is not enabled by default and the `set-monitoring-console-configuration` is used to enable it on demand.

To enable the GUI manually run the following asadmin command:

```
asadmin> set-monitoring-console-configuration --enabled=true
```

Payara InSight is available at `http://localhost:8080/monitoring-console/` . Like the Admin console's user interface, it only runs on the DAS.

To stop Payara Insight run this command:

```
asadmin> set-monitoring-console-configuration --enabled=false
```

TIP	Stopping the application does not disable the data collection. This is a background process that is controlled separately as described in the next section.

Enabling Data Collection

The collection of monitoring data gets enabled and disabled in connection with the general configuration: *Monitoring* → *Monitoring Service* → *Enabled*. By default, this setting is enabled.

NOTE	If you want to remove the monitoring data collection from the server's runtime without disabling the monitoring service itself you must delete the `monitoring-console-core.jar` from the `as-install/modules` folder of your server installation and restart the server.

Enabling Data Aggregation (History)

By default, InSight has a sliding window of data per metric of usually one minute length which contains 1 point per second. This allows typical "real-time" monitoring. These points represent the raw data as it is probed/collected.

Without a history points that get "pushed" out of the sliding window are "forgotten".

A history can be enabled which aggregates the points per seconds to 1 aggregated point per minute, the minutes to 1 point per hour, the hours to 1 point per day. These aggregations equally have sliding windows, so they hold the recent hour, day or month worth of data.

The longer points are in the past the fewer points per fixed time-frame there are:

- In the most recent minute there are the raw (not aggregated) points per second.
- In the most recent hour there is 1 point per minute from aggregating the seconds within the last minute.

- In the most recent day there is 1 point per hour from aggregating the minutes within the last hour.
- In the most recent month there is 1 point per day from aggregating the hours within the last day.

The aggregation of points must be enabled using the following Asadmin command:

```
asadmin> set-monitoring-console-configuration --history-enabled=true
```

This command starts the aggregation which increases over time. The aggregated data is only stored in memory. Widgets can choose to show this data using the *Extras - History* setting.

The aggregation of points can be disabled again using the following asadmin command:

```
asadmin> set-monitoring-console-configuration --history-enabled=false
```

NOTE All aggregated data is discarded and cannot be restored. Equally, history is lost when the server is restarted.

The aggregated history feature is designed to keep memory usage fairly low. 1000 metrics require in the range of single digit MBs of memory.

In typical environments the extra memory and CPU effort for the history should not be an issue.

Graphical User Interface

After enabling Payara InSight open the home page at `/monitoring-console/` (usually on port `8080`).

The GUI ships with a set of preconfigured pages each focussing on a different aspect of the server.

NOTE	Some metrics are only available when the monitoring configuration of certain server parts is set to `HIGH`. Widgets that do depend on such a configuration changes let you know how to enable the metrics.

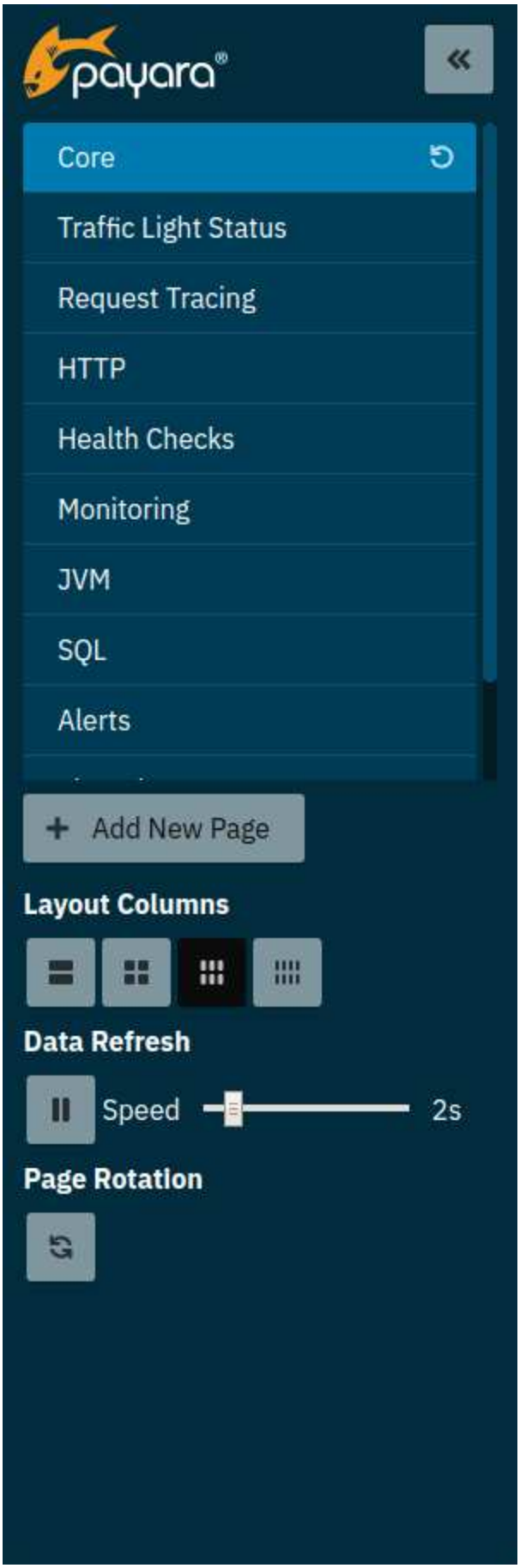
Pages

Payara InSight is organized in fully customizable pages. A page is a collection of widgets arranged in columns. Usually, pages are focused on visualizing metrics of a specific area of the server, like HTTP or request tracing.

The pre-configured pages are a starting point that can be changed and extended by the user. New custom pages can be created and populated with arbitrary metrics.

Switching Pages

To switch page use the left menu.


payara
Core
Traffic Light Status
Request Tracing
HTTP
Health Checks
Monitoring
JVM
SQL
Alerts
Add New Page
Layout Columns
Data Refresh
Speed
2s
Page Rotation

Core

TIP	Each page switch also updates the URL making them "bookmarkeable". URLs can be copied and shared with others to share particular pages with them.

Updating URLs after switching pages also means that once the page is changed using the dropdown navigation, the browser's page history based navigation can be used to go back to already visited pages.

Changing Page Layout

Each page uses a column layout of one or more columns. The number of columns for a page can be changed using one of the *Layout Columns* quick access buttons.

Changing the column layout of a page does not change the preferred position of the widgets. If a widget is set to column 3 but a 2-column layout is used all widgets of the 3rd or 4th column get pushed to the 2nd column instead. This allows switching layouts based on the viewing device without redoing the configuration entirely.

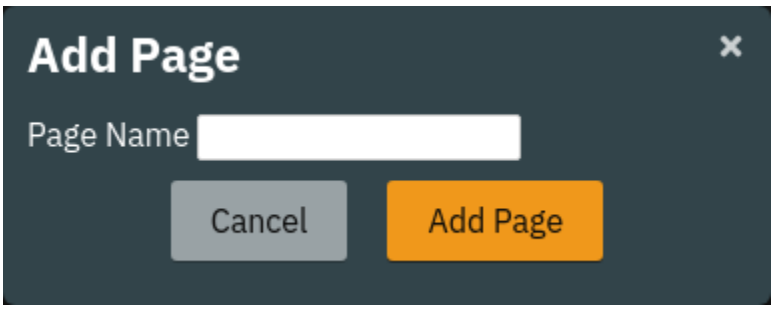
Changing Page Update Interval

By default, the graphs on a page update every 2 seconds. To control the interval use quick access on the left side menu or *General* application settings.

The interval can be paused and continued without changing the interval duration by using the *play/pause* toggle next to the interval setting.

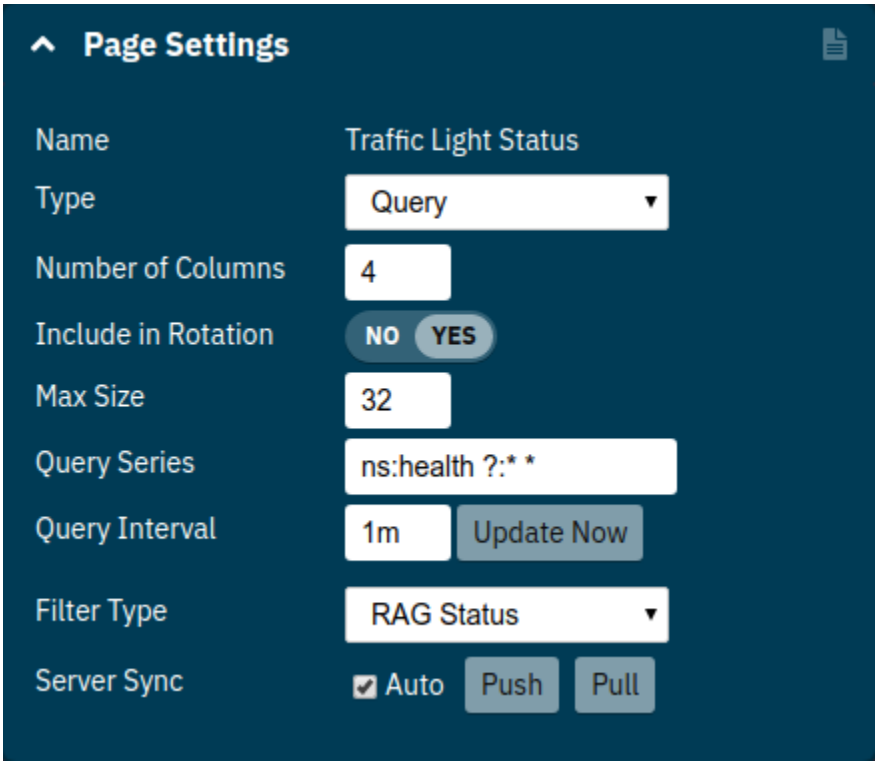
Adding Pages

To add a new page expand the left menu, click the *Add New Page* button below the list of pages and enter a name for the new page in the dialogue.



Usually pages are populated manually. Use the *Add New Widgets* button in the right sidebar (*+* when minimized) to populate the page and configure the added widgets.

Alternatively pages can use a query to automatically populate the page with widgets for those metrics that match the query. To use a query page open the page settings and switch the *Type* to *Query*:



Fill in *Query Series* and *Query Interval* and click the *Update Now* button.

NOTE	Query pages are repopulated in intervals. Any changes to individual

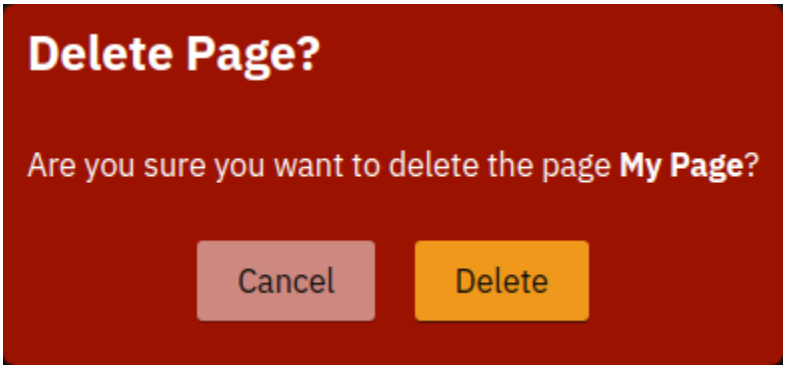
> widget settings will be overridden with the next population. However, queries can be used to initially populate a page and then switch the page back to *Type: Manual* so that changes to the widgets are kept. Naturally this comes at the downside that metrics no longer are added to the page automatically.

Learn more about series patterns as used by query pages.

Deleting a Page

To delete a page open the left sidebar and click the on the trash icon next to the page name in the list of pages you want to delete.

Confirm the deletion.

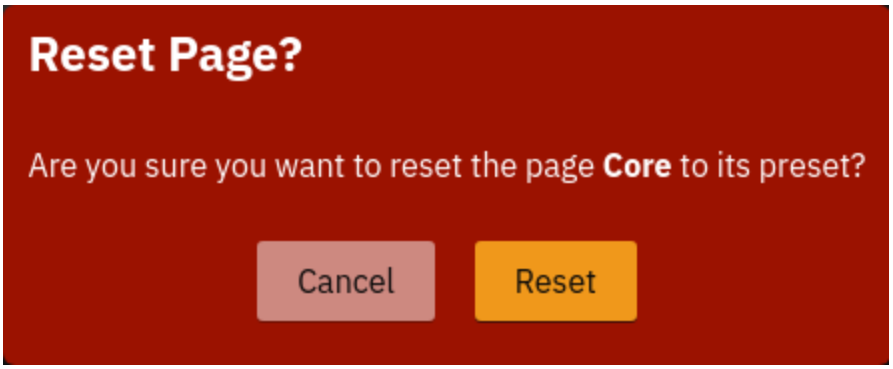


NOTE Pages that are based on a preset cannot be deleted. They can only be reset to their preset configuration.

Reset a Page

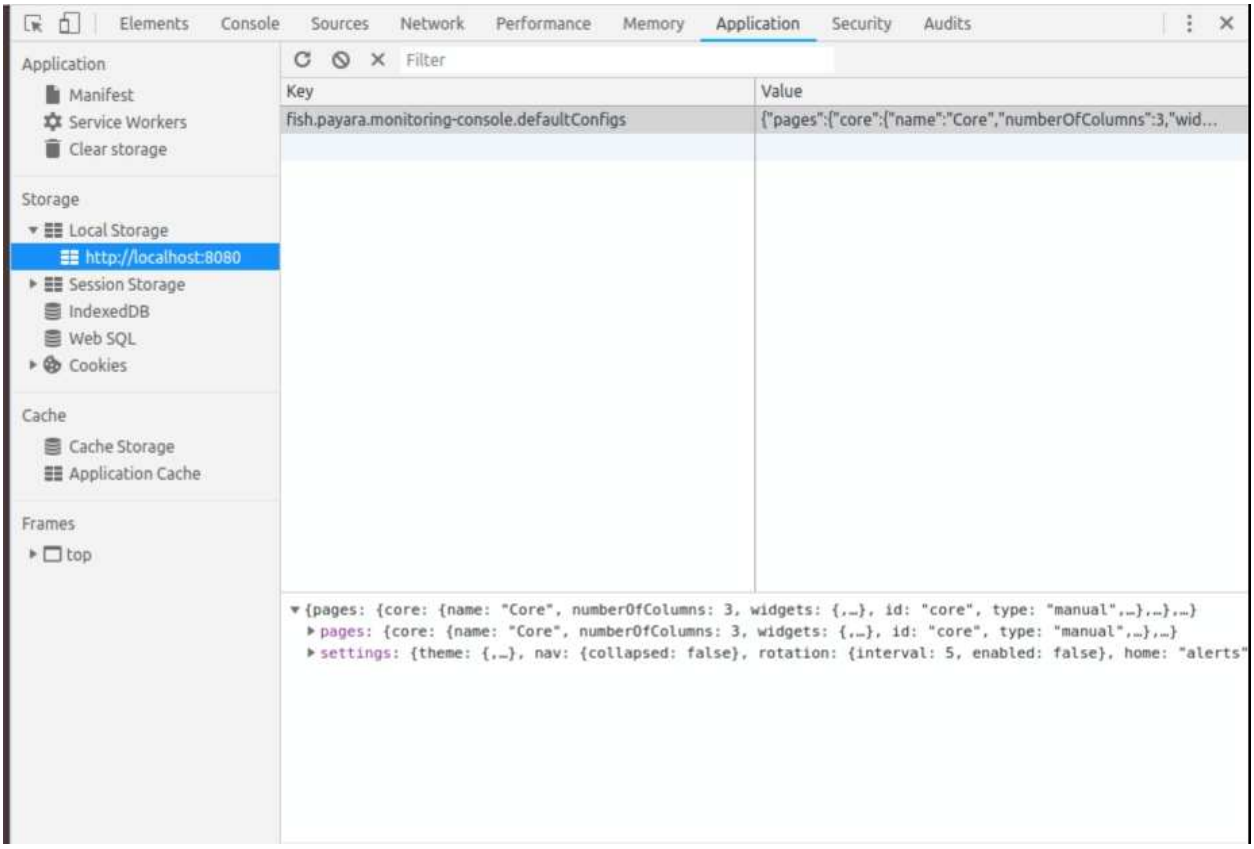
When the configuration of a preconfigured page or any of the widgets on it was changed by the user the page can be reset by clicking the *Reset* icon next to the page name in the expanded left side menu.

Confirm the reset.



WARNING	The reset will affect all widgets and the general page settings of the currently shown page. All changes made previously by a user will be overridden.

NOTE	To reset all pages and remove all user-created pages, clear the browser's local storage of Payara InSight.



Page Rotation

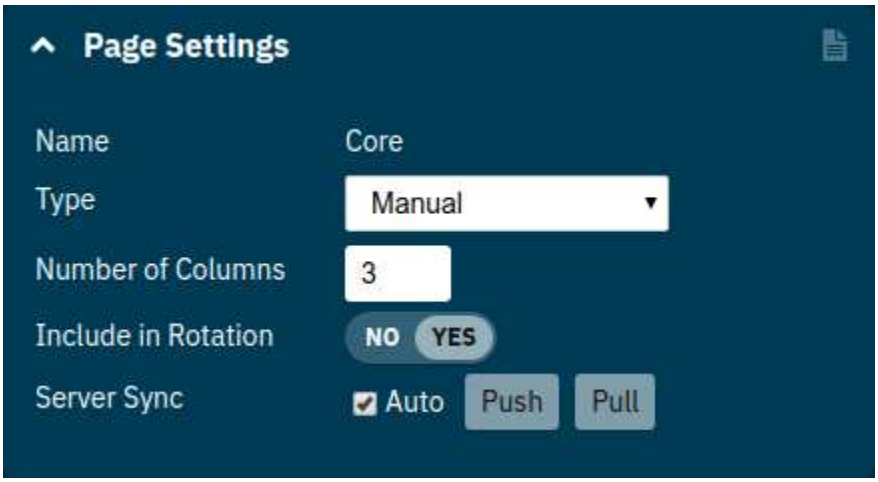
Payara InSight is meant to be used to display metrics on status monitors. To allow

viewing a variety of pages and their metrics on a single screen the pages can be automatically rotated showing each page for a configured interval duration.

To start or stop page rotation either use the quick access button at the button of the left side menu or use the toggle switch in the *General* application settings. This starts or stops the rotation immediately and each page is shown for the configured duration starting with the currently shown page.

The duration can be adjusted by changing the text field of the *Page Rotation* to another time value. Use units to express the duration, as for example: `1min`, `30sec`. Changes take effect immediately.

Individual pages can be excluded from rotation by toggling the *Include In Rotation* toggle to *No* in the page settings.



Page Management

Payara InSight uses the browser's local storage to store the page configuration (which includes the widgets placed on a page). This means by default changes to page configurations do not affect other users.

To make pages available to others and to start from a common base configuration pages are explicitly synchronised with a persistent server page configuration.

The monitoring console can be used in one of three roles:

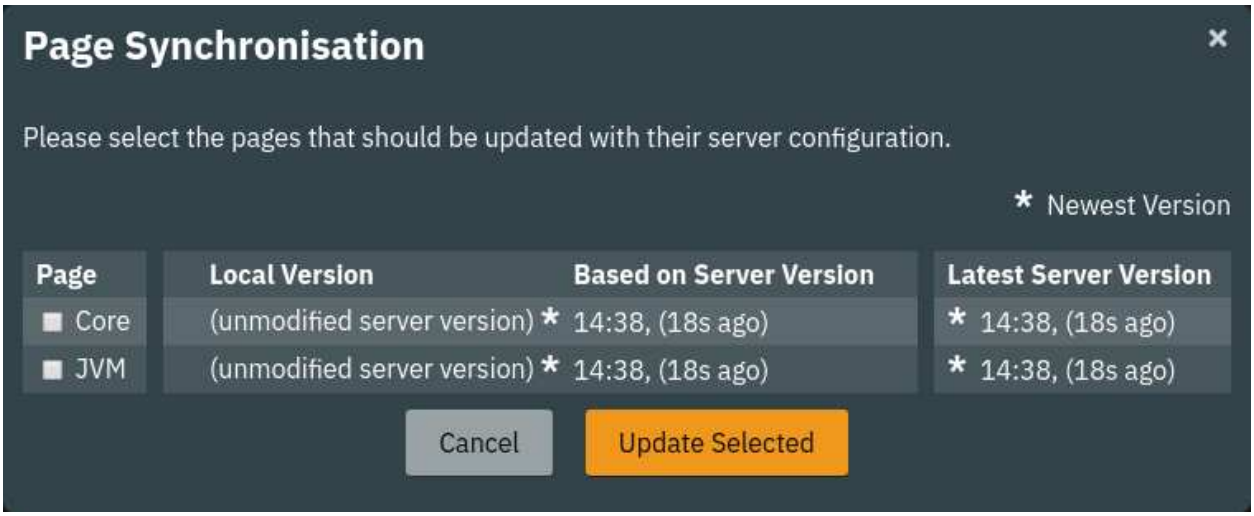
- *Guest*: A user that uses Payara Insight only on occasion to check existing metrics on existing pages ("read-only" user).
- *User*: A user that uses the console to experiment with its features without affecting others.
- *Administrator*: A user that uses the console to configure the configuration that is the basis for other users.

The chosen role controls the behaviour and available options for the page synchronisation:

Administrator: This role that is meant to manage the persistent server page configuration. The *Administrator* pushes local changes to the server to update the base configuration. ***User***: ***Guest***: They pull the server configuration to update their local configuration with the base.

The control remains with each user.

The *Page Synchronisation* dialogue shown when entering the GUI allows the user to pick the pages that should be updated (pulled).



By default, all pages available remotely that have not been changed locally are preselected.

NOTE	*Guests* automatically pull the servers base configuration, no dialogue is

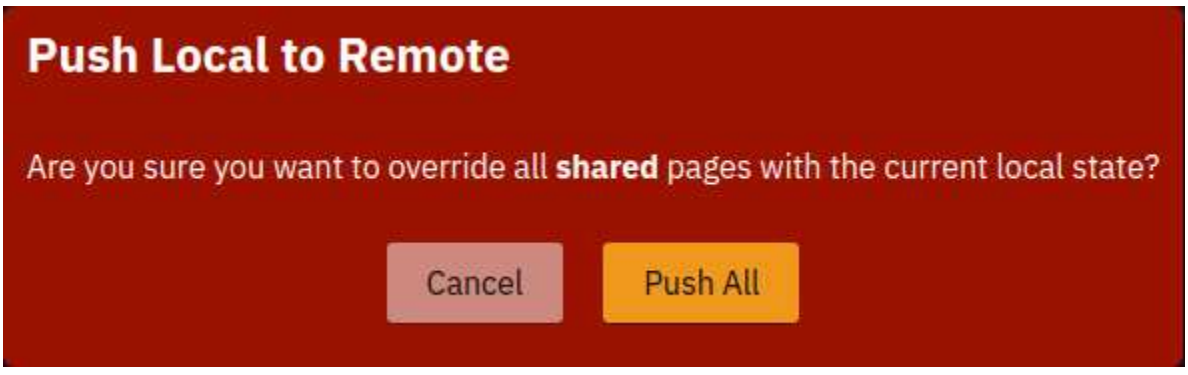
> used

The newest version is marked with an asterisk for easier orientation.

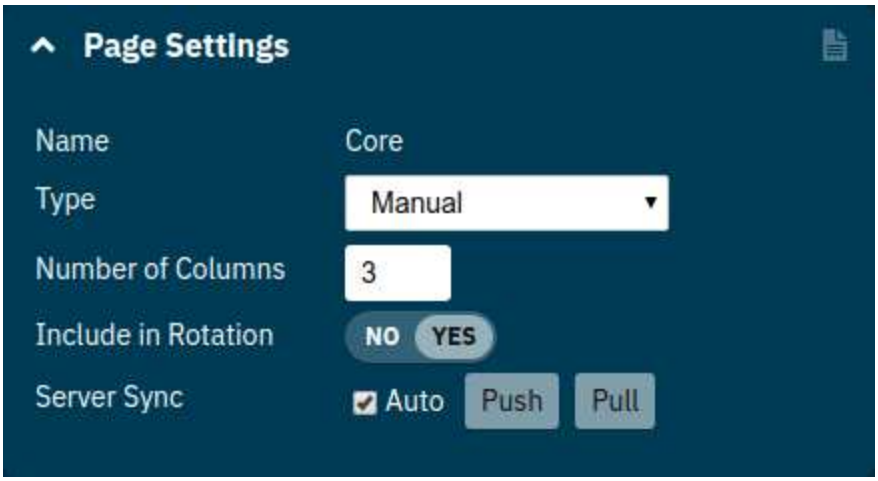
This dialogue can also be opened from the *Pages* application settings by clicking the *Update Local…* button for the *Page Sync* property.

Administrators also find a button to *Update Server…* pages with any changes applied to local changes. This is only needed for pages that are not using automatic updates in their page settings.

This simply should give administrators different approaches to managing the configuration of shared pages. To bulk update all server pages click the *Update Server…* button and confirm the update:

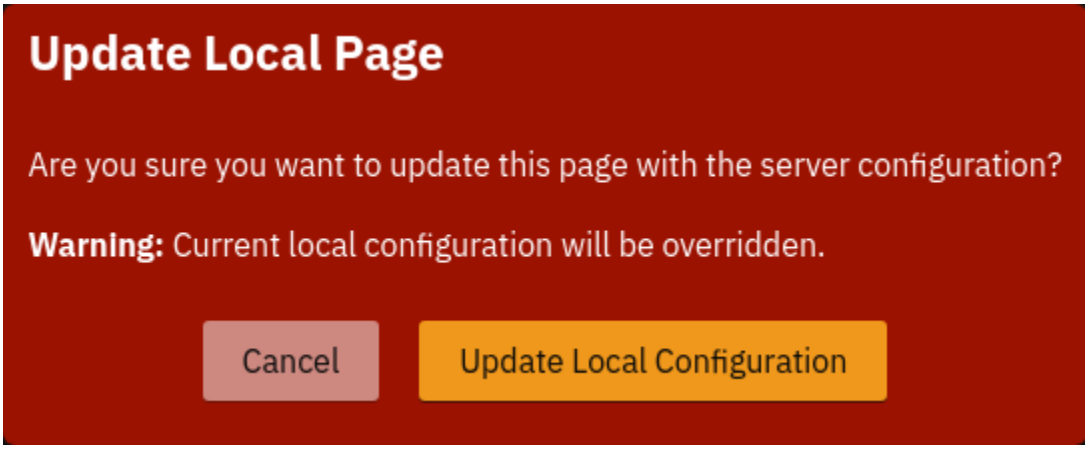


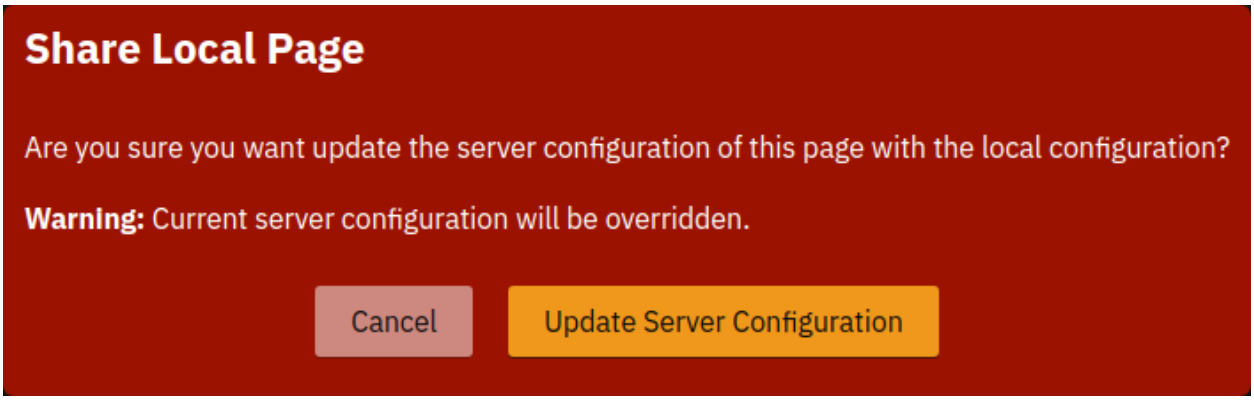
To share a page that so far is not part of the persistent server configuration open the *Page* settings and look for the *Server Sync* property.



Users and *Administrators* can both *Push* a page to share or update the persistent server configuration or *Pull* the server configuration to update the local page. *Administrators* can also enable or disable the *automatic* page synchronisation. When enabled any local change done by an *Administrator* is automatically pushed to the remote configuration.

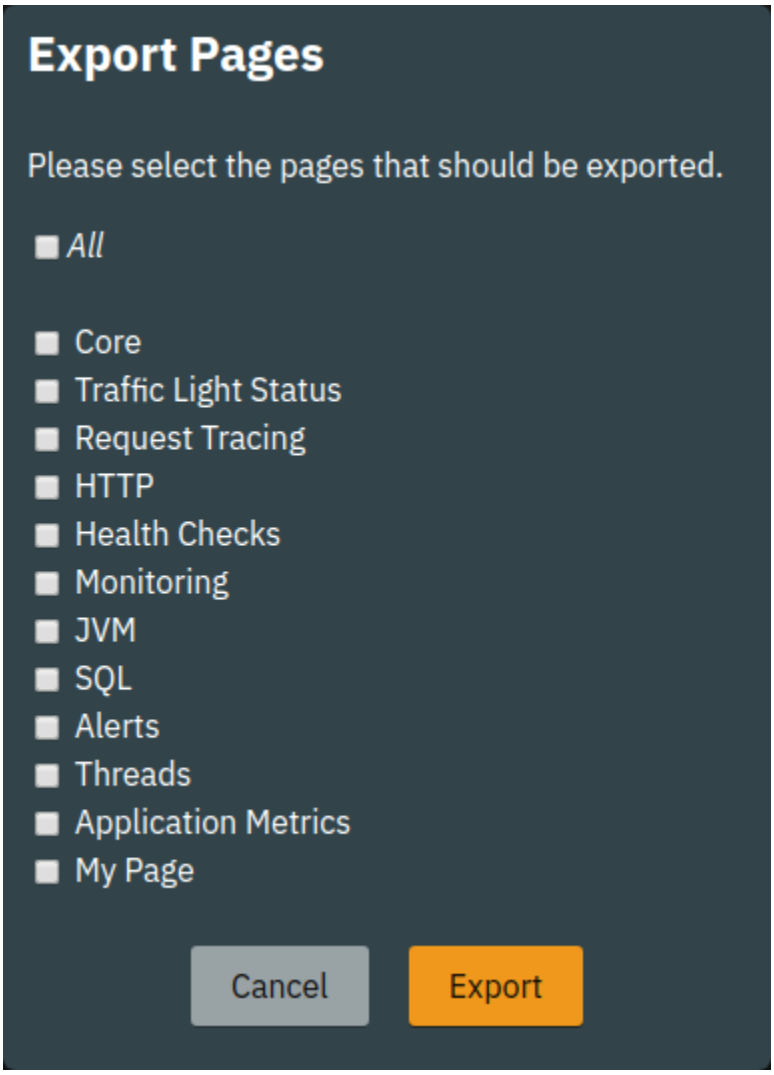
Both *Push* and *Pull* need to be confirmed.





Page Export

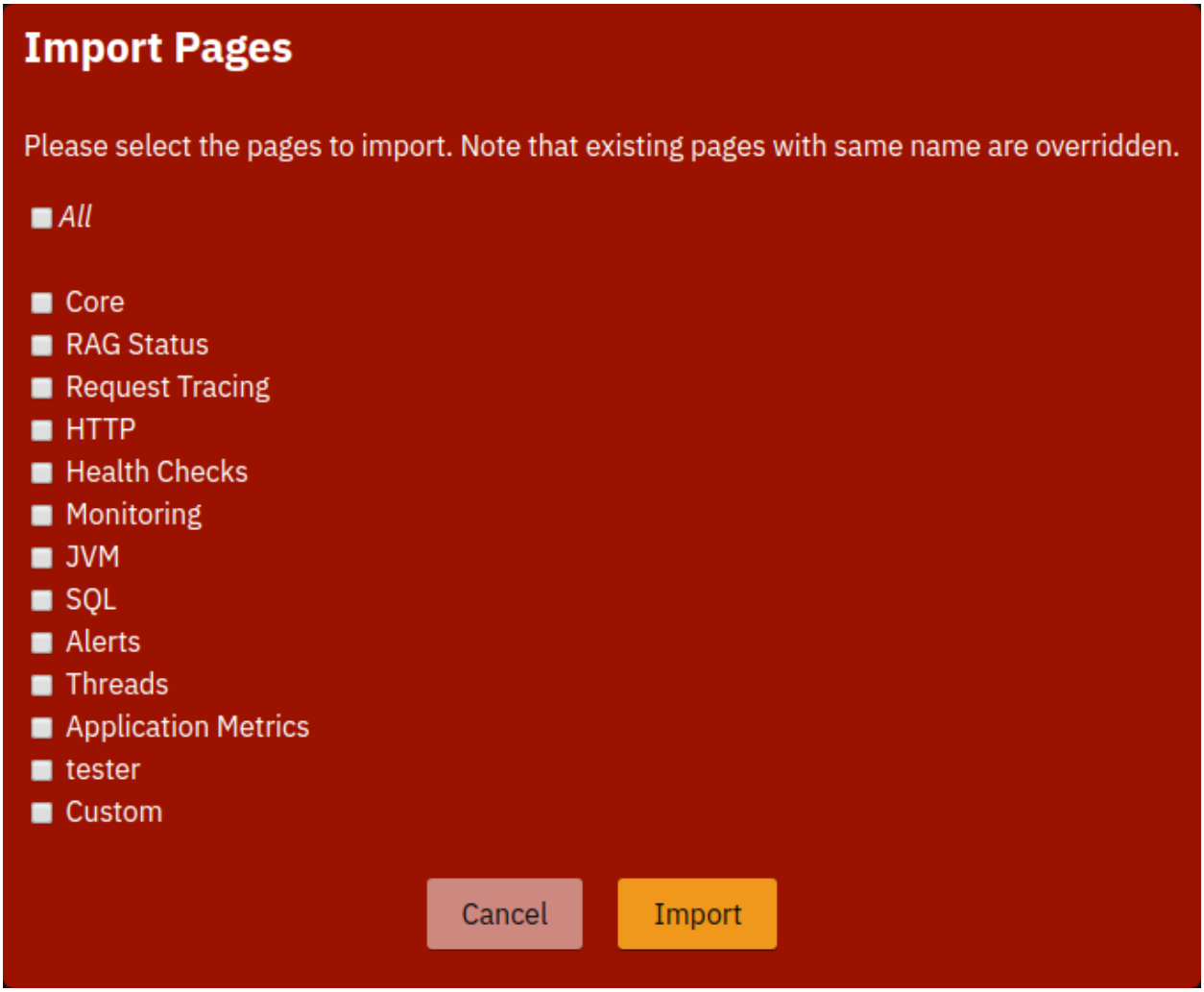
Besides the managed page synchronisation which is using the persistent server configuration the local configuration can also be managed manually. To manually export pages click the *Export…* button and select the set of pages to export in the dialogue and confirm your selection by pressing *Export*



Page Import

To manually import pages click the *Import…* button and select a file. The dialogue will show the list of pages contained in the file.

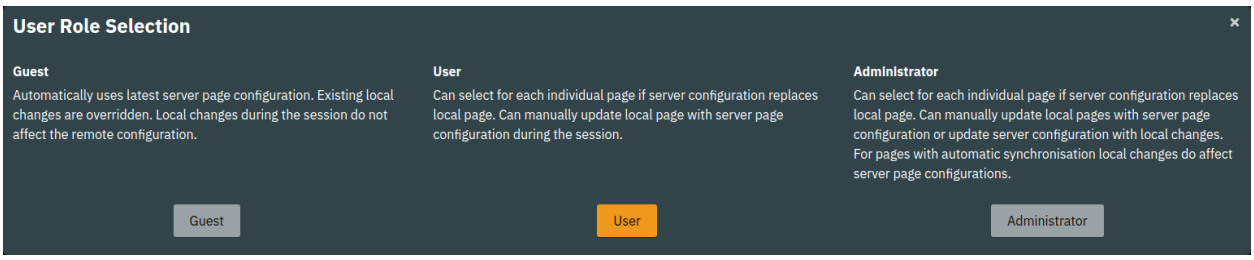
Select the pages that should be imported:



WARNING	Importing a page will override the local page with the same ID (name). Any customisation made to the local page will be lost.

Changing User Role

When no role is set from previous sessions the user is asked to choose a role when opening the Payara InSight application.



The selection can be changed at any time in the *General* application settings for *Role*.

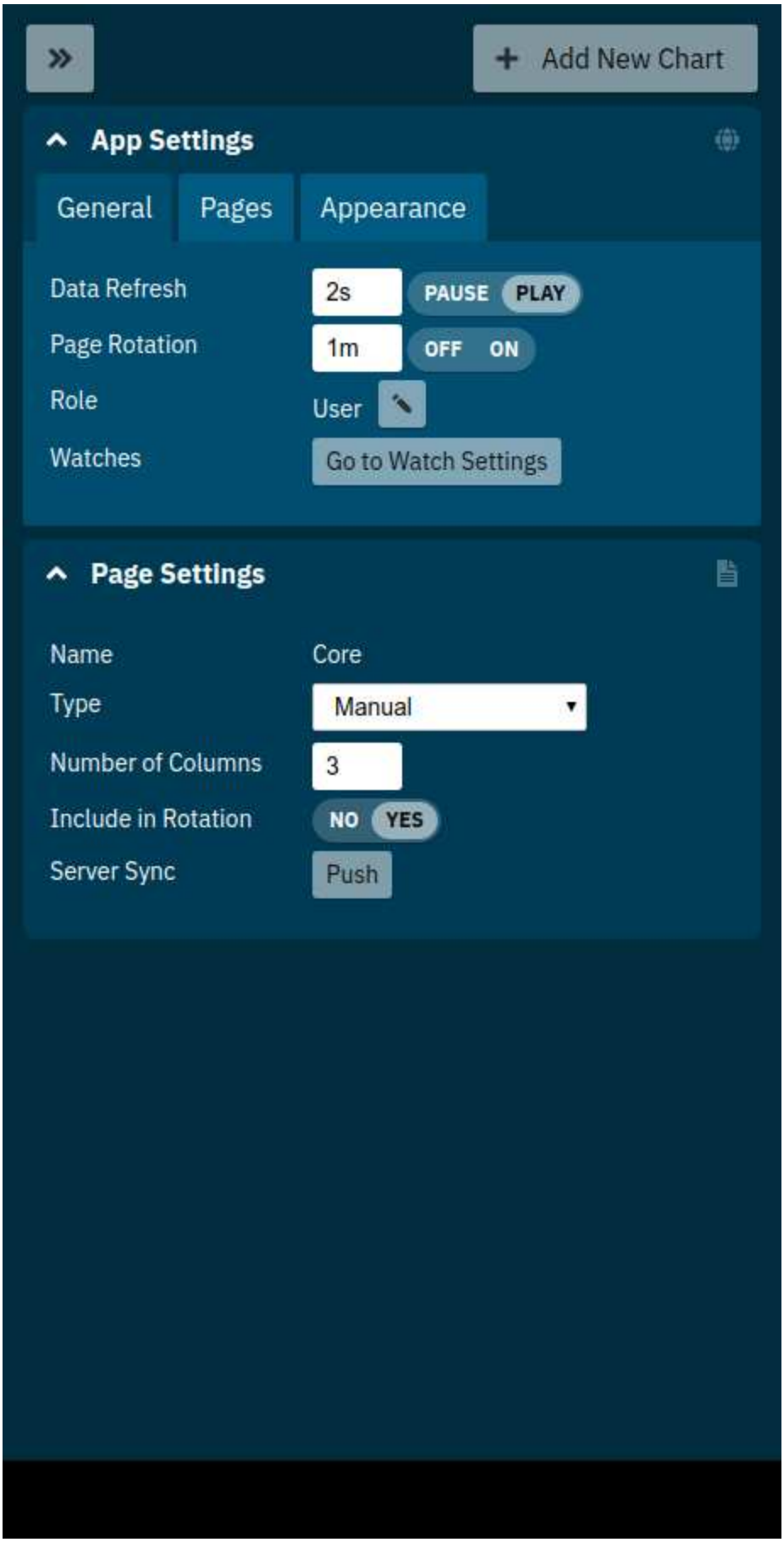
NOTE	Roles are not connected to accounts or other forms of permission management. Users can switch between the roles to suit their current intentions when using Payara InSight.

Widgets

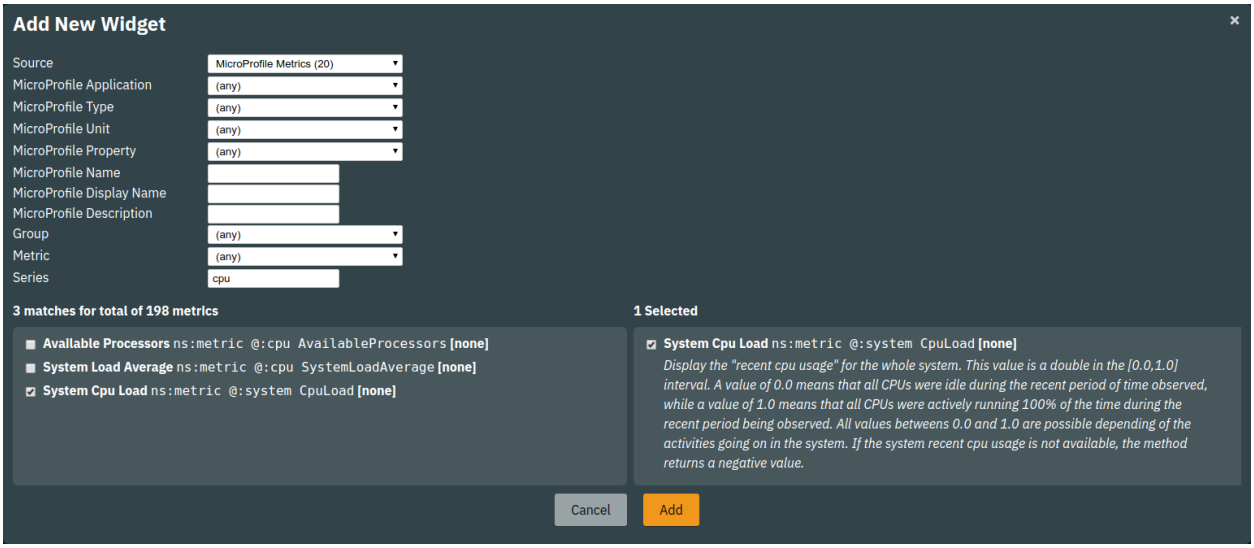
Widgets are the components that can be placed on a page to show a particular metric or a set of metrics. While there are different types of widgets usually a widget shows a line graph of one or more metrics for the entire cluster.

Adding a Widget

Widgets can be added to any page. Click the plus button in the minimized right side menu or the *Add New Chart* button when it is expanded:


Add New Chart
App Settings
General
Pages
Appearance
Data Refresh
2s
PAUSE
PLAY
Page Rotation
1m
OFF
ON
Role
User
Watches
Go to Watch Settings
Page Settings
Name
Core
Type
Manual
Number of Columns
3
Include in Rotation
NO
YES
Server Sync
Push

This opens the *Add New Widget* dialogue.



Use any one of combination of the filter dropdowns to select a subset of metrics and/or enter a query text in any of the free text search input fields to further narrow down the results. All matching metrics will be shown in the list on the left.

Select one or more items. Selected metrics are shown in the list on the right. The search can be changed between selections.

Finally, confirm the selection by clicking the *Add* button of the dialogue.

Newly added widgets are automatically placed in a column so that widgets are evenly distributed on the page.

Use the side panel widget settings to change the column, width, height or item position of the widget or use the quick menu for the widget by clicking the pencil symbol in the top right corner of the widget.

Configuring Widgets

To configure a widget select it by clicking on its title or the pencil icon in the top right corner of the widget.

This opens the right settings side panel and shows the detailed configuration of the widget. The side panel can be closed by either deselecting the widget, by clicking on its

title again, or by clicking the collapse icon of the right sidebar.

NOTE	If the settings have been opened explicitly through the *Settings* menu an empty selection will not close the settings side panel.

The *General* widget settings are concerned with the widget's position within the page and its general appearance.

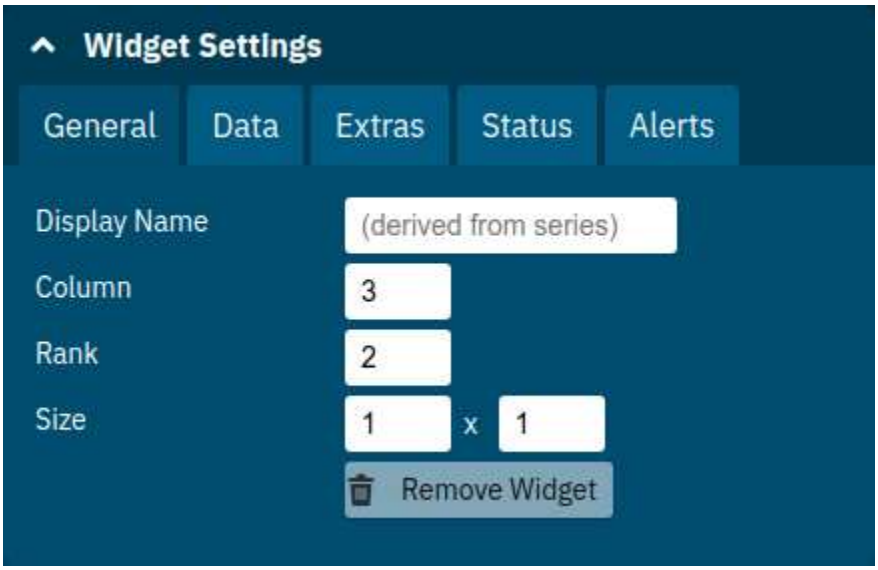


Table 167. Widget General Settings

Property	Default	Description
Display Name	`(undefined)`	Optional name to customise the displayed widget title. If no name is given the *Data* ▯ *Series* is shown.
Column	1	The column in the layout the widget **prefers** to be in.
Rank	1	Within columns widgets are sorted by their item value, lowest values first. The item position is automatically updated during page layout should it be needed.
Size	1 / 1	How many width (number of columns) and height (number of rows) the widget should span. Note that this is a target size.

The *Data* settings are concerned with what and how the data is shown in the graph of a widget.

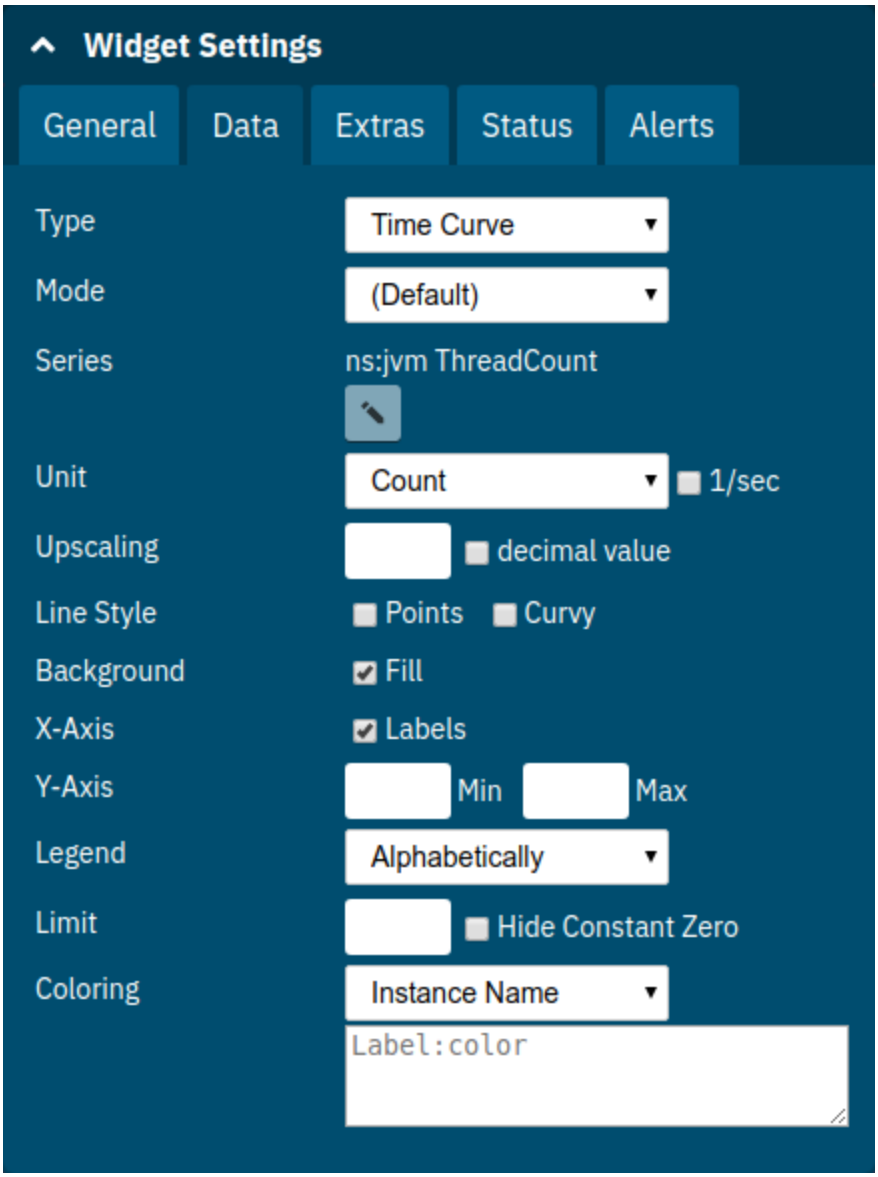


Table 168. Widget Data Settings

Property	Default	Description
Type	`Time Curve`	Changes the type of graph used to show the metric. This also affects which data of the metric a widget is focussed on. Possible choices:
Mode	*(Default)*	If available for the type of widget the display mode can be switched between *Table* and *List*. Table view is best fitted for annotations that share common attributes, lists are better fitted for annotations with potentially differing attributes.

Property	Default	Description
Series	-	The metric key as given when adding the metric, use pencil icon to edit
Unit	`Count`	The unit controls how the raw number of a metric is interpreted and represented e.g. when displaying in axis labels. The unit also controls what values are accepted as input for numbers, like thresholds, that are on the same axis. Possible units are: `Count`, `Milliseconds`, `Nanoseconds`, `Bytes`, `Percentage` and `Up/Down`.
Unit 1/sec	`false`	Whether to show the metric as a change per second (delta between two points in the series normalised to average delta per second).
Upscaling	`(undefined)`	When defined all values are multiplied by this factor. This can e.g. be used to move a decimal range between zero and one to a percentage range `0-100`.
Upscaling *decimal value*	`false`	Check to mark the metric as one that is provided as a decimal value during collection (floats or doubles and alike). This will then revert the conversion applied during collection which converts decimal values to an integer number with 4 available decimal digits by multiplying them by `10,000`. This divides values by 10,000 again to move them back into their original value range.
Line Style	`(none checked)`	Options on how to display the current value line in the graph: Check *Points* to show individual data points on the line, check *Curvy* to use Bézier curves instead of straight line connections.
Background	*Fill*	When *Fill* is checked the area between line and axis is filled. The background transparency can be configured globally in the *Appearance* application settings.
X-Axis	*Labels*	Whether to show labels for the x-axis.
Y-Axis	`(undefined)`	Set a minimum or maximum number for the y-axis instead. When not set range is derived automatically from the data points.

Property	Default	Description
Legend	*Alphabetical*	The order applied to the legend. Possible choices: `None`: Items are shown as send by the server, `Alphabetically`: Items are shown in alphabetical order (instance name first, series based label second), `Increasing Value`: Items are shown ordered from smallest to largest current value, `Decreasing Value`: Items are shown ordered from largest to smallest current value,
Limit	(undefined)	When set the maximum number of legend items shown (especially useful when ordering legend items)
Limit Hide Constant Zero	`false`	When set series that have been zero for a while are not shown in legend and line graph
Coloring	*Instance Name*	Used to select the colours the data is shown with. *Instance Name* will show data in the same colour that belongs to the same instance. *Series Name* shows the metric series in the same colour (independent of instance). This is individual to each widget. *Result Set Index* will show each matching series in a different colour. *Instance and Series Name* shows the same instance and series in the same colour. This is global. This means another widget using this colouring with the same instance and series will have the same colour as well.
Coloring Mapping	(undefined)	A space separated mapping of series group values to a named colour value as defined by the application appearance *Defaults* settings. For example to color a series shown with label *AmberAck* in amber color the entry is `AmberAck:amber` (`amber` here refers to the default name used for *Degraded* (internal name)).

Configuring Widget Extras

The *Extras* settings are concerned with additional information shown in or around the graph or widget content in general.

This includes visual helpers that can be added to a graph of a widget in order to make it

easier to for the user to quickly understand the data displayed.

In particular these are reference lines and threshold indications.

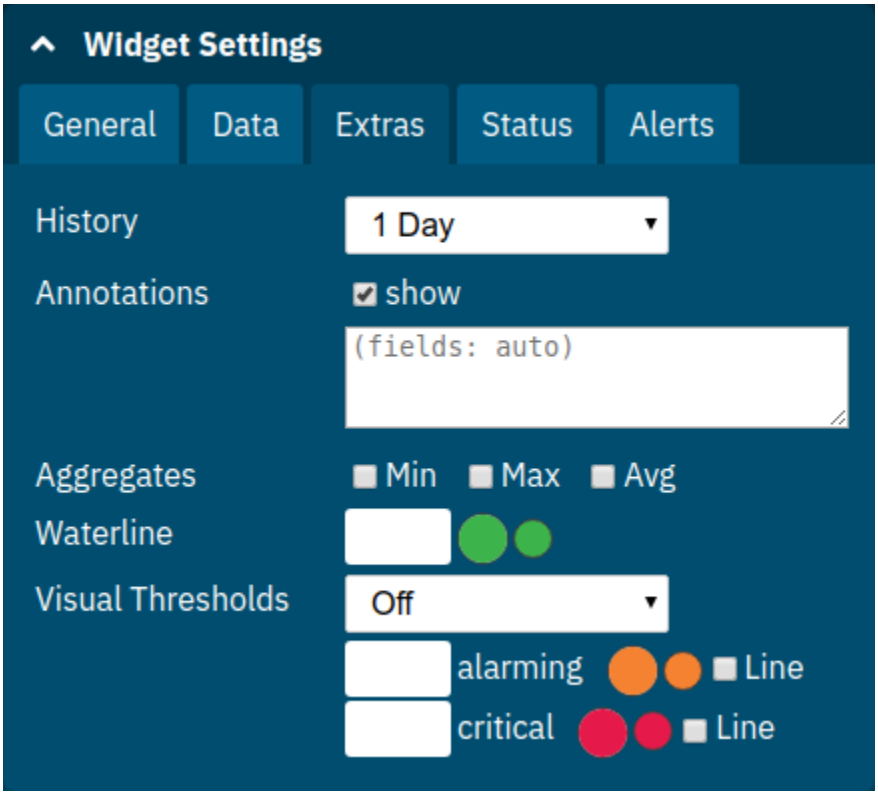


Table 169. Widget Extras Settings

Property	Default	Description
History	*None*	What period of additional aggregated data to show in the graph: • *None* shows just the recent minute ("real-time") • *1 Hour* adds 1 point per minute in the recent hour (the most recent minute continues to show 1 point per second) • *1 Day* adds 1 point per hour in the recent day (the recent hour continues to show 1 point per minute, the recent minute 1 per second) • *1 Month* adds 1 point per day in the recent 31 days (again recent day continues 1 point per hour, recent hour per minute and recent minute per second). **NOTE** This data is only available after enabling the history aggregation using the asadmin command.
Annotations	(checked)	Whether to show annotations for in an alert table.
Annotations (Fields)	*(blank)*	A comma or space separated list of fields to show for an annotation list or table. The given field order is reflected in the view.
Aggregates	(none checked)	Check to add the all time minimum, maximum or average line to the graph
Waterline	(undefined)	Adds a simple reference line that could mark an upper or lower limit. Enter in unit of y-axis, e.g. 20ms, 5s, 2GB, 4%, 123
Threshold Reference	`Off`	Controls what value the *Alarming Threshold* and *Critical Threshold* are compared to. Usually this is the most recent value but in some cases the all time minimum, maximum or average are useful too.

Property	Default	Description
Alarming Threshold	(undefined)	The limit for the *"Alarming"* state. If *Critical Threshold* is undefined or if its value is larger than this value the alarming threshold is exceeded if the reference value is larger than the threshold. Otherwise, if the *Critical Threshold* is lower than the *Alarming Threshold*, the threshold becomes 'inverse', and is exceeded when the reference value falls below the alarming value. When exceeded the status becomes *"Alarming"*. Enter in unit of y-axis, e.g. `20ms, 5s, 2GB, 4%, 123`
Critical Threshold	(undefined)	The limit of the *"Critical"* state. If *Alarming Threshold* is undefined or its value is smaller than this value the critical threshold is exceeded if the reference value is larger than the threshold. Otherwise, if the *Critical Threshold* is lower than the *Alarming Threshold*, the threshold becomes 'inverse', and is exceeded when the reference value falls below the critical value. When exceeded the status becomes *"Critical"*, overrides status *"Alarming"*. Enter in unit of y-axis, e.g. `20ms, 5s, 2GB, 4%, 123`

Besides adding a reference line to the graph the critical and alarming thresholds affect the status evaluation.

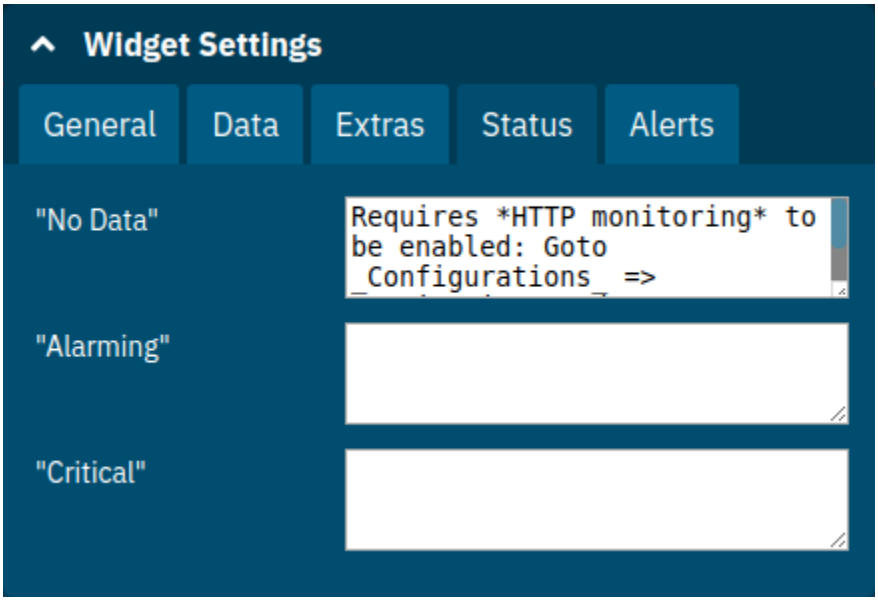
Use the colour input to individually override the widget's colour for waterline, alarming line or critical line in graphs.

Press the button right next to it which shows the default colour as configured in the *Appearance* application settings to reset the individual colour to use the default setting instead. A colour set to the default changes with the default.

Configuring Widget Status Messages

The current value of a metric can automatically be categorised as a certain status.

For example based on the threshold values of decorations or by encountered errors or missing data. The *Status* settings allows to set a custom message for a particular status.



Property	Default	Description
`No Data`	`(undefined)`	The message to show when no data for a metric is available. For example to hint about configurations needed to make the metric available.
`Alarming`	`(undefined)`	The message to show when the threshold got exceeded and the status evaluates to *Alarming*.
`Critical`	`(undefined)`	The message to show when the threshold got exceeded and the status evaluates to *Critical*.

The messages support simple markup using `*bold*` and `_italic_`.

Configuring Widget Alerts

The values of metric series can be tracked by the server to report values that out of a healthy range.

This is represented by states:

- *Red*: values in this range indicate an unhealthy state
- *Amber*: values in this range indicate a degraded state
- *Green*: values in this range indicate a healthy state
- *White*: values in this range are in no particular state (neither good nor bad, maybe undecidable)

State changes to amber or red are reported as alerts. This is shown in the widget they affect.

The settings allow filtering these related alerts to prevent them from showing. This makes sense on pages that have dedicated widgets to show alerts or just if users prefer to see the graph instead.

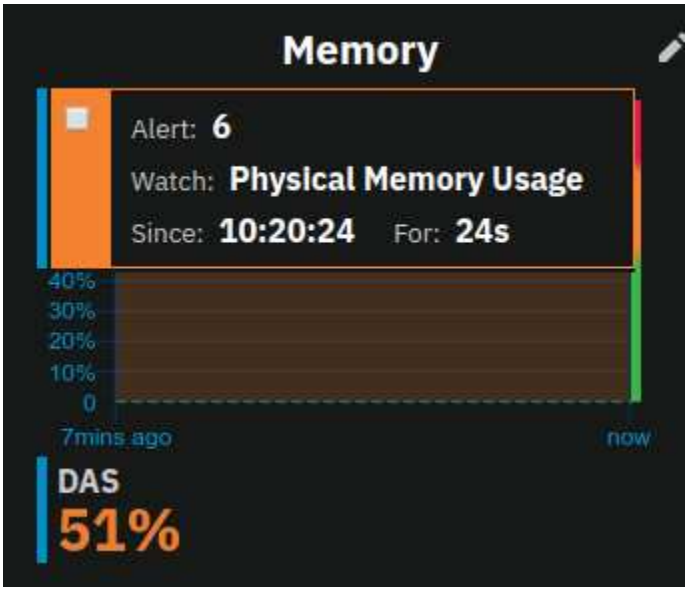
Property	Default	Description
Filter	(none checked)	Do not show (when checked) *Amber* and/or *Red* alerts, *Ongoing* and/or *Stopped* alerts, *Acknowledged* and/or *Unacknowledged* alerts.

Handling Alerts

Alerts can occur when a watch is configured for a metric series.

A watch describes the conditions and thresholds used to determine a state from the recent values of a metric series. It is either *Red* (unhealthy), *Amber* (degraded), *Green* (healthy) or *White* (undetermined).

A *Red* or *Amber* state causes an alert that is shown on top of the graph.

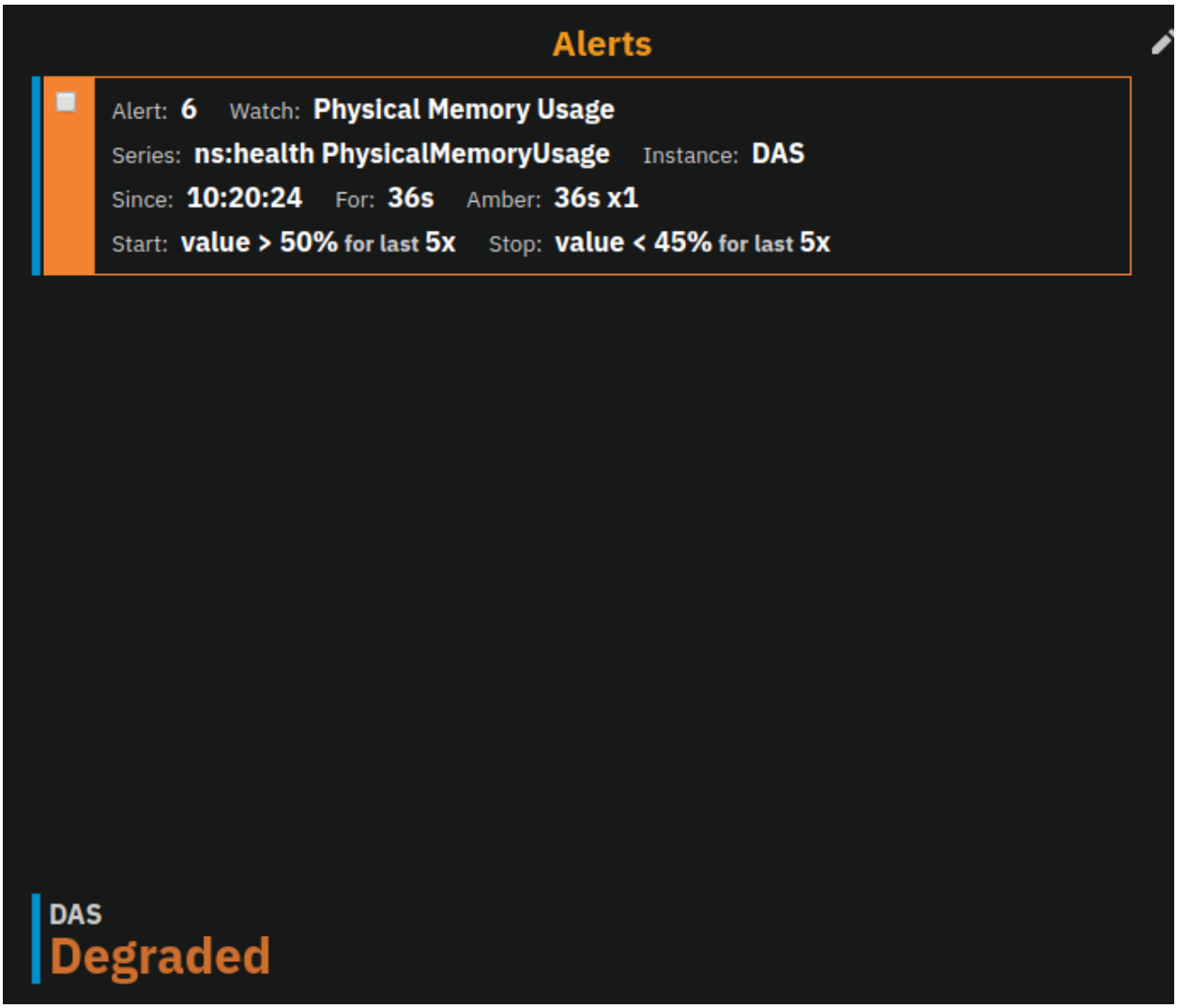


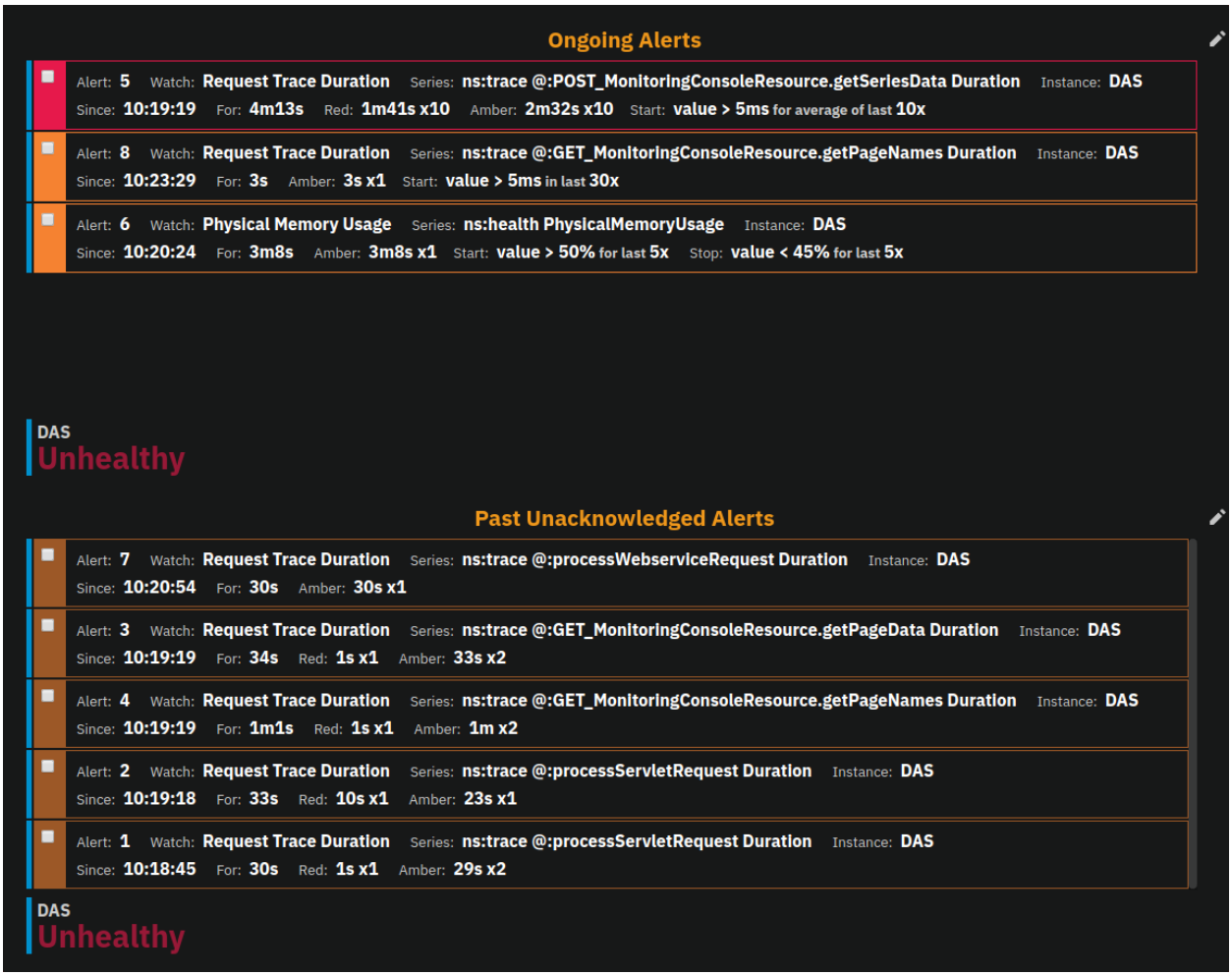
When an alert occurred for a metric shown in a graph the background of the line causing the alert is coloured in *Red* or *Amber* default colour until the alert is stopped.

Alerts are considered ongoing as long as the state is *Red* or *Amber*. If it is evaluated to *Green* or *White* ("unspecified") the alert stops. Stopped or acknowledged alerts are no longer shown on top of line graphs.

Other ways to hide an alert from a line graph is to acknowledge the alert by checking its checkbox or to configure the widget's *Alerts Filter* setting so that alerts generally aren't shown for that widget.

To see stopped or acknowledged alerts add a widget of type *Alert Table*.


Alerts
Alert: 6 Watch: Physical Memory Usage
Series: ns:health PhysicalMemoryUsage Instance: DAS
Since: 10:20:24 For: 36s Amber: 36s x1
Start: value > 50% for last 5x Stop: value < 45% for last 5x
DAS
Degraded



By default, an alert table shows all alerts including stopped and acknowledged alerts. Use the widget *Alerts* settings to exclude alerts with certain properties.

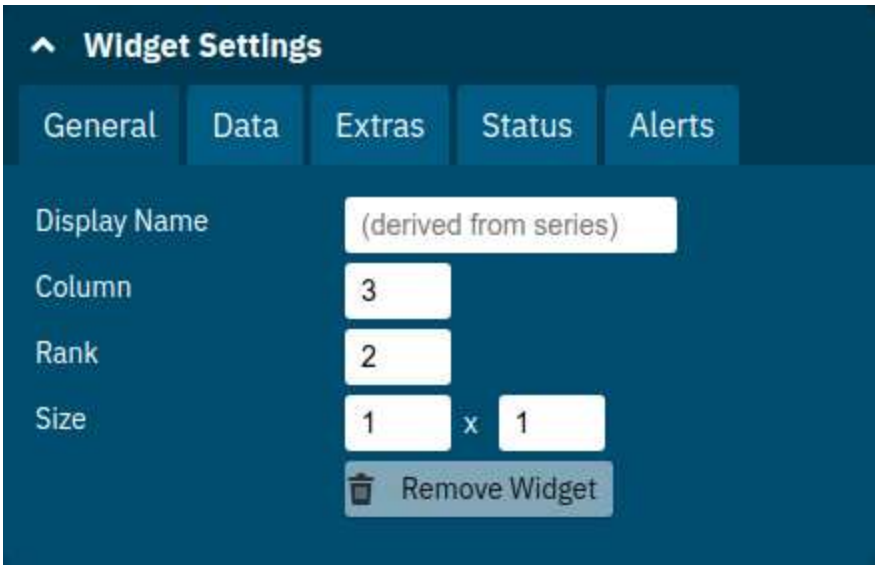
An alert table shows the overall state of each instance in the table's legend.

As mentioned before the *Alerts* settings can equally be used for line graphs to create a more restrictive setting.

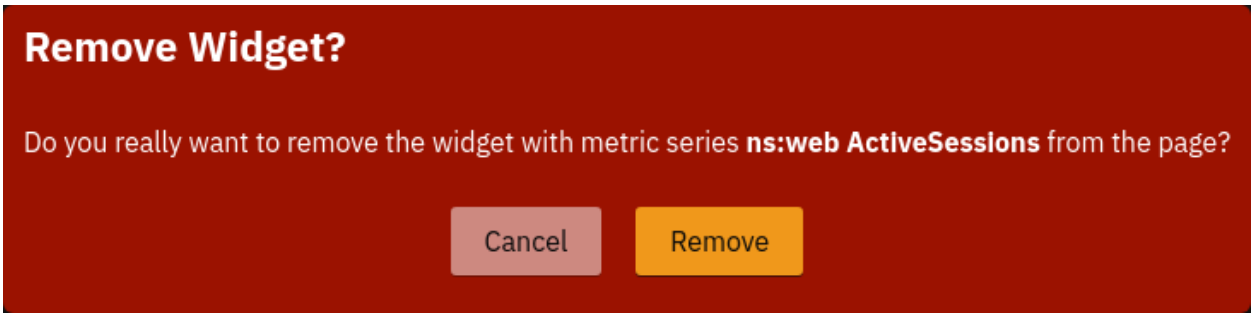
Removing a Widget

Any widget can be removed from any page. Click the pencil symbol in the top right of the widget or its title to open the widget's settings.

On the *General* widget settings tab you find the *Remove* button.



Confirm you want to remove the widget.



Remember that a page originating from a preset can be reset to recover a widget that was removed accidentally.

Common Page Settings

Most of the page settings have already been described in context of the different page related user actions. The following tables are included for reference.

The common page settings are those used by pages of *Type Manual* and *Query*.

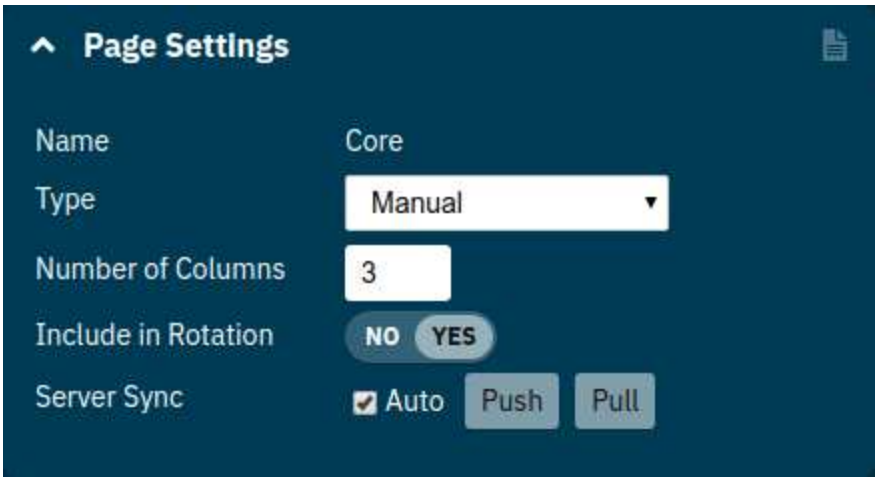


Table 170. Common Page Settings

Property	Default	Description
Name	-	The name as given by the user when the page was created or as defined in the page preset, only user-created pages can be renamed
Type	`Manual`	• `Manual`: user populates pages with widget using add widget • `Query`: a query is used to populate page in intervals
Number Of Columns	1	The number of columns available to place widgets in
Include In Rotation	Yes	• Yes: Page is included in the set of pages rotated during page rotation • No: page is excluded from this set
Fill Empty Cells	No	• Yes: empty cells in widget grid are filled by extending the span of the widget above • No: empty cells left empty shown with a button to add a widget
Server Sync	(checked)	`Auto`: Changes made by an administrator to the page or the widgets of this page are automatically pushed to the shared configuration if and only if the page has been explicitly pushed to the shared configuration at some point in the past.

NOTE Pages of *Type: Query* have more settings to configure the contents of the page.

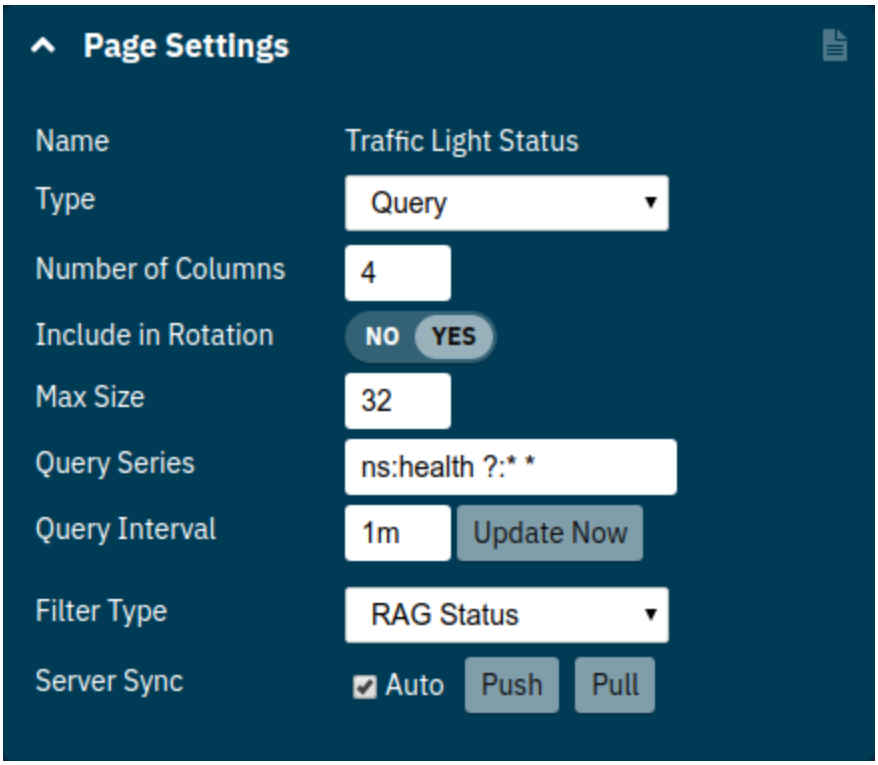


Table 171. Query Page Settings

Property	Default	Description
Max Size	`32`	Maximum number of widgets added to the page when populating it from query matches (even if there are more metrics matching the query)
Query Series	-	A series pattern used to query for metrics, matches are used to populate the page
Query Interval	`1m`	The interval in which the query is run and the page is repopulated with matches
Filter Type	-	When selecting a type only metrics inferred to prefer that type are shown on the page

See Application Metrics page for an example of a query page.

Global Configurations

The configuration that controls global properties of the Payara InSight user interface. It is stored in the browser's local storage and thereby individual for each user.

NOTE | These settings cannot be shared.

General Application Settings

Table 172. Application General Settings

Property	Default	Description
Data Refresh	`2s`	Interval duration used to update the currently shown page with new data (poll duration). To pause data update toggle the *play/pause* toggle switch.
Page Rotation	`1m`	Duration each page is shown during a page rotation. To run or stop the rotation toggle the *on/off* toggle switch. See section Page Rotation for details.
Role	*User*	Select the used user role. See section Changing User Role for details.
Watches	-	Click the *Go to Watch Settings* button to open the Watch Management

Pages Settings

The *Pages* application settings give access to different page management operations as described in detain in section Page Management

Alerts Settings

The *Alerts* application settings contain general alert configuration that applies globally (to any page).

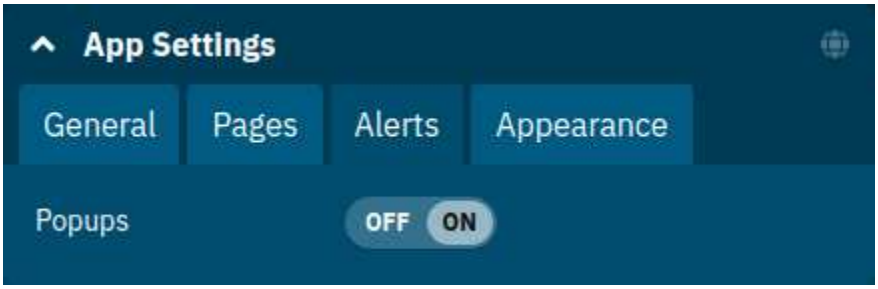


Table 173. Alerts Settings

Property	Default	Description
Popups	*On*	When *On* any alert status change is shown in form of a pop-up dialogue regardless of the currently viewed page.

Appearance Settings

Payara InSight allows you to configure the colours used in graphs. There is a global colour configuration affecting all graphs.

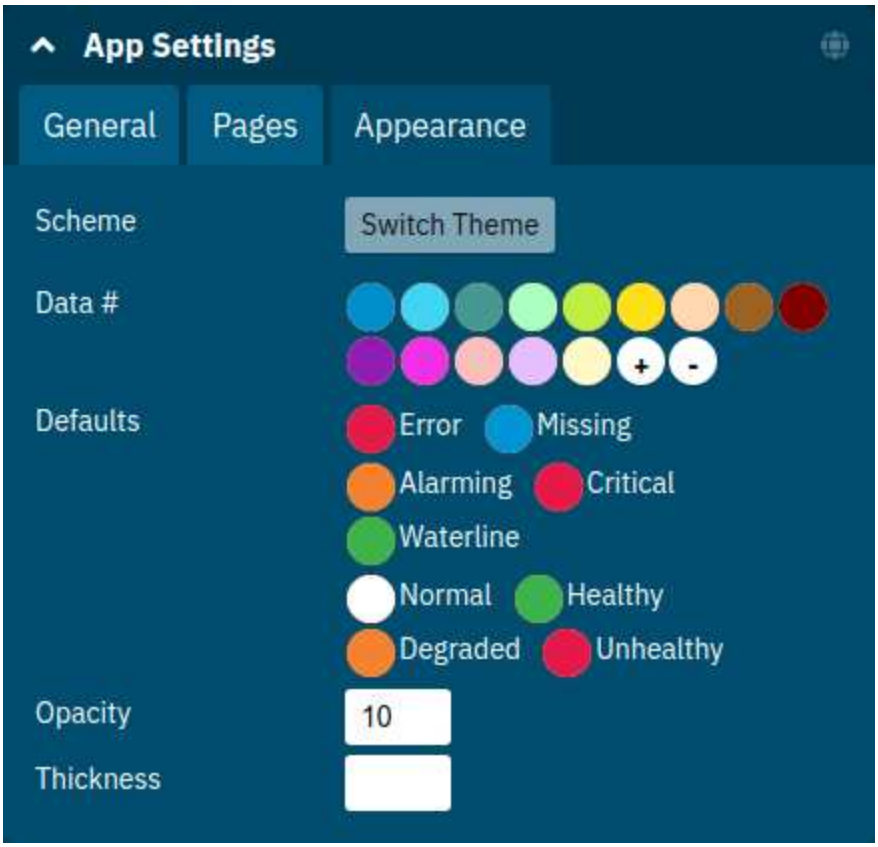
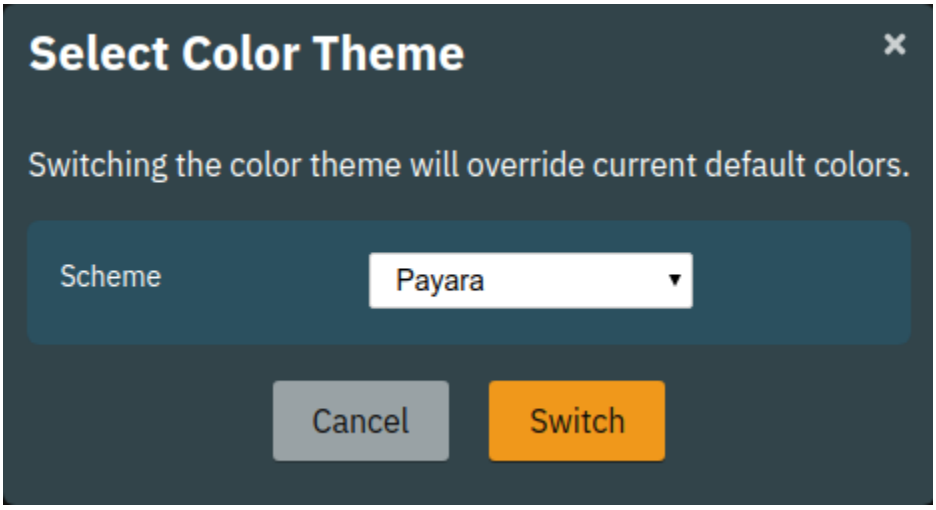


Table 174. Appearance Settings

Property	Default	Description
Scheme	(Payara Branding)	The selected scheme is applied to set the colour palette for *Data* as well as *Defaults* and *Opacity*.
Data # (Palette)	(colours of *Payara Branding* scheme)	The list of colours to use when colouring data series in graphs. This is the line colour in line charts or the bar colour in bar charts. Background will use the same colour with *Opacity*. Add new colours by using the `+` button. The `–` button removes the last colour of the list. If there are fewer colours in this palette than needed for different lines in a graph new colours are derived based on this palette. The derived colours focus on bright colours with high contrast but might not always fit well with the rest of the palette. In such case consider adding more colours to the palette.

Property	Default	Description
Defaults	(default of *Payara* scheme)	Sets the default colours for *Waterline*, *Alarming* and *Critical* as well as *Red*, *Amber*, *Green* and *White* state lines, backgrounds and texts. Some of these can be individually overridden in the widget's configuration for these lines.
Opacity	10	Background opacity for lines or bars in graphs, 0 (fully transparent) to 100 (solid).
Thickness	(3)	Line width on a scale `1–8`. Each point corresponds to 0.5px with. Default for lines in a line graph is 3, so 1.5px.

Themes act as a (p)reset of the data colour palette, default colours and *Opacity* setting. When switching theme these settings are reset to the theme's value:



Any further customisation of individual colours can and should be done after applying the theme of choice as switching theme would reset such customisations again.

Alerts Management

The values of series can be monitored using watches. If values fulfil all aspects of a start condition of a watch an alert is created.

While alerts transition between the states *Unhealthy*, *Degraded*, *Healthy* and possibly *Normal* based on the watch definition they originated from there are two manual status

fields available for each alert:

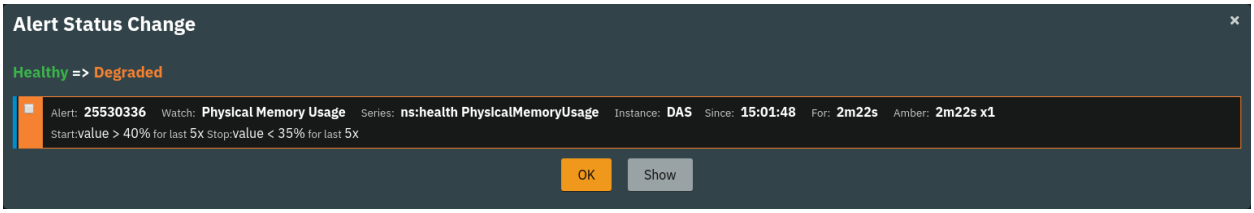
- **Global**: *Acknowledged*/*Unacknowledged*: When an alert transitions to a more severe state it resets to *Unacknowledged* waiting for users to manually acknowledge that they took notice (and care) of the degrading state transition.
- **User**: *Confirmed*/*Unconfirmed*: Alerts are unconfirmed for a user if that user has not seen and confirmed the alert transition yet.

Both of these manual states should help users to keep track of alert transitions and the alerts they have dealt with.

Acknowledge Alerts

Alerts are manually acknowledged by system operators when they have checked and reacted to the alert and consider it done. It is a form of alert management tool that is shared by all users.

To acknowledge alerts click the checkbox on left of an alert row.

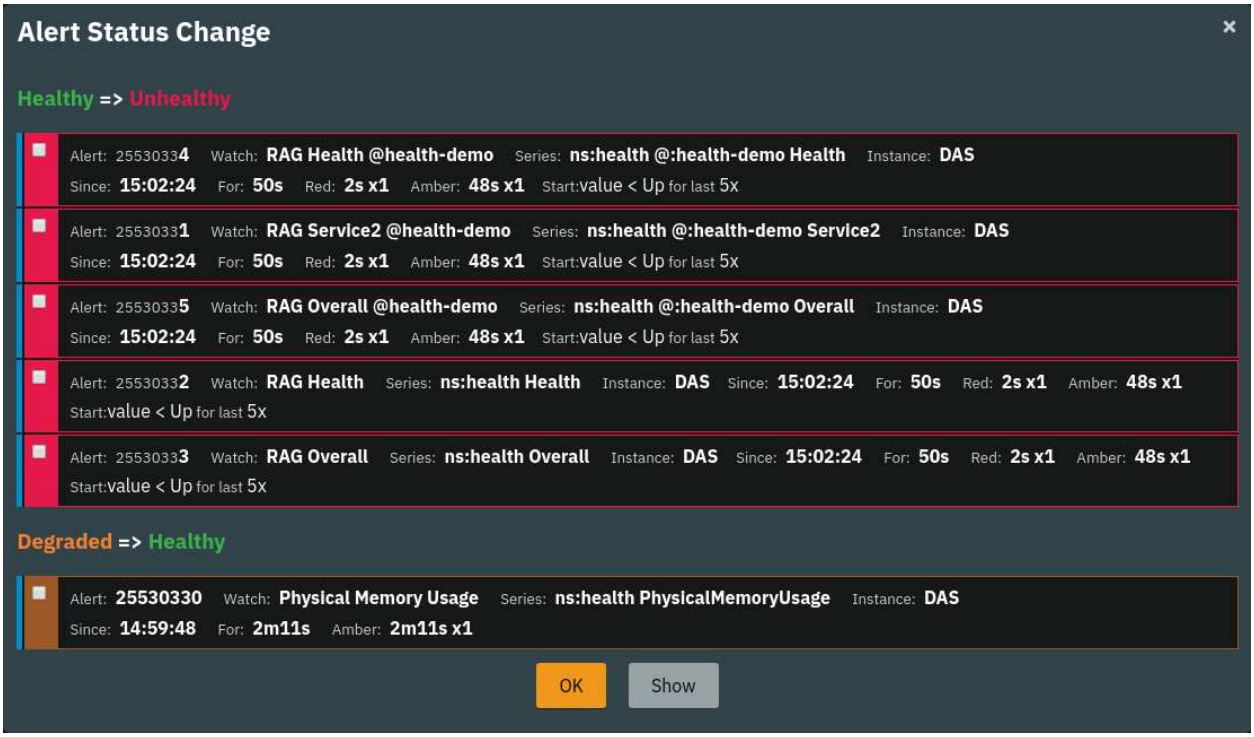


An alert can be acknowledged:

- When shown as part of the *Alert Status Change* pop-up dialogue
- When shown on top of a line graph widget
- When shown as part of an alert table widget

Confirming Alerts

When the state of alerts changed compared to the last state that has been confirmed by a user the change is indicated by the *Alert Status Change* pop-up dialogue. It shows all transitions that took place since the user last confirmed alert status changes.



When a user confirmed the change by either pressing *OK* or *Show* the shown transitions are considered as confirmed and are no longer shown.

NOTE	The confirmation status is individual for the user (browser local state) so each user has its own confirmed state and based on his/her last confirmation will get different updates.

The confirmation pop-ups can be disabled in the Application Alerts Popups setting. When disabled transitions are implicitly confirmed until the setting is enabled again. This is a more modest setting where alerts are only indicated globally by the alert indicator in the footer.

Alert Status Indicator

All alerts are indicated in the bottom right corner of the footer so that users do not miss alerts that would otherwise not show on the current page.

The indicator gives a summary of the ongoing alerts:

- In Red: # of Unacknowledged *Unhealthy* state alerts (# of acknowledged *Unhealthy* state alerts)
- In Amber: # of Unacknowledged *Degraded* state alerts (# of acknowledged *Degraded* state alerts)

Click on the indicator to jump to the alerts overview page.

Watch Management

A watch describes the conditions to meet to start and stop alerts in relation to a specific metric. For example to get an alert every time an HTTP request causes a server error (responds with any of the `5xx` status codes) a custom watch is installed.

This is very similar to a custom health check that could be linked to any metric available where the thresholds are defined as part of the watch. There are many varieties of conditions that can be formulated. Each watch has a start condition which when met starts a new alert.

Optionally a stop condition can be given which when met will stop the alert. If no such stop condition is given an alert stops as soon as its start condition is no longer met.

Such pairs of start and stop conditions can be defined for 3 levels:

Unhealthy: corresponds to a "red" alert **Degraded**: corresponds to an "amber" alert **Healthy**: conditions for particularly good values, these will not cause alerts but can be used to mark a *healthy* range in graphs.

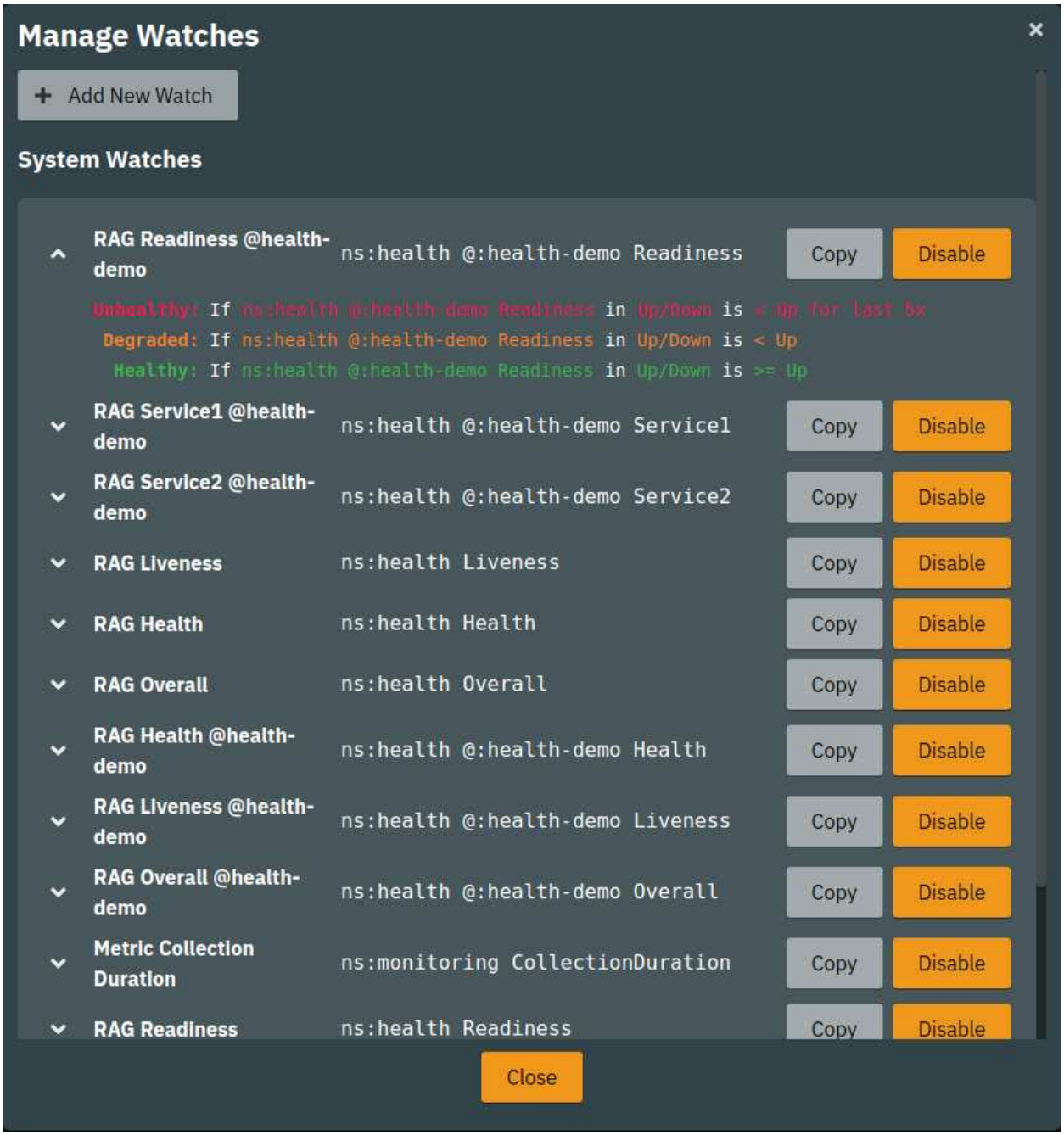
Each watch must have at least an *Unhealthy* or a *Degraded* condition. When both are given alerts can transition between *Degraded* and *Unhealthy* as the same issue or alert.

Unhealthy takes precedence over *Degraded* and *Healthy*, *Degraded* takes precedence over *Healthy* should their condition be met at the same time.

Should a *Degraded* alert be acknowledged, but it transitions to *Unhealthy* afterward the acknowledgement is reset so this change is not missed.

To open the watch management click the *Go to Watch Settings* button in the *General* application settings:

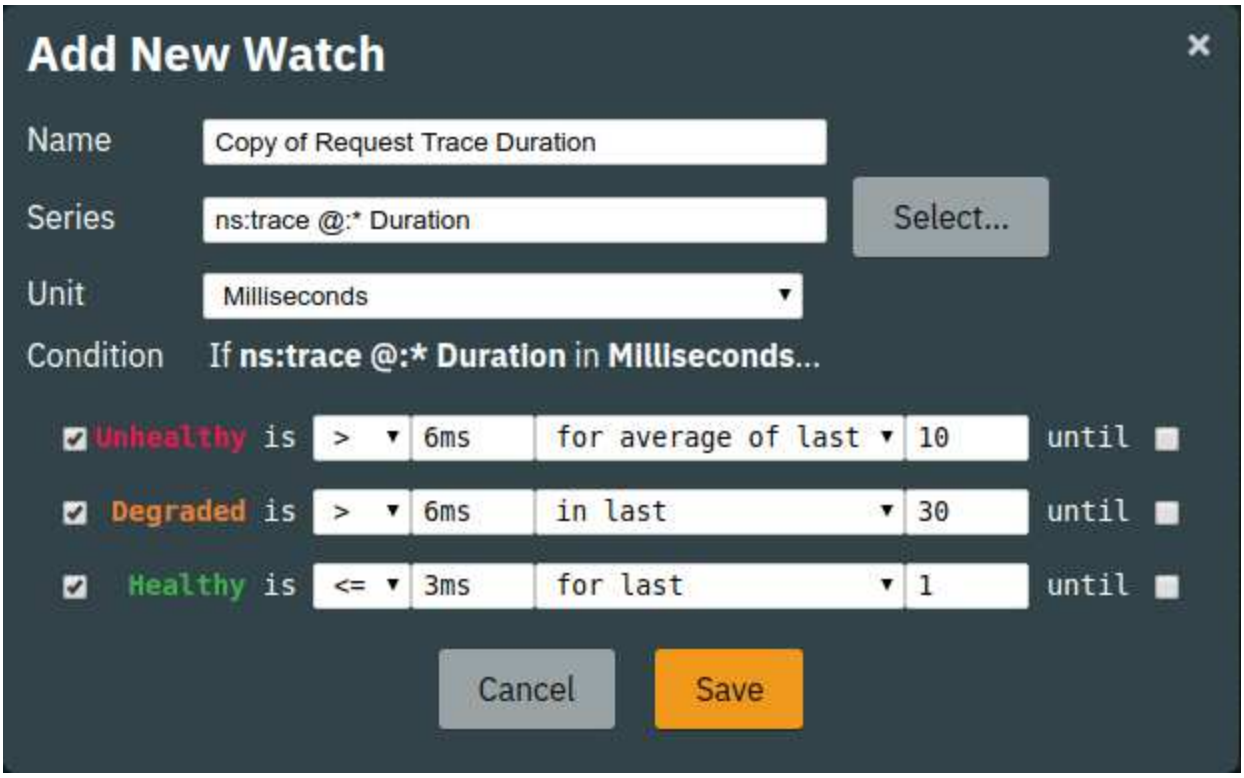
NOTE	Generally the appearance of the watch management and the available options are dependent on the user role. Full access in given to *Administrators*. *Users* are restricted to viewing and creating new user watches. *Guests* can only view the settings.



The list shows *System Watches* and *User Watches* (below if any exists). To expand and see a watches' conditions click on the down arrow or its name to toggle the details.

Creating a Watch

Open the watch management as described above and click the *Add New Watch* button at the top.



The above example shows a relatively simple watch first copied from an existing watch which then is customised. The name of the watch can be anything as long as it is unique.

Each watch should have either an *Unhealthy* and/or a *Degraded* level. It is also recommended to add a *Healthy* level. Any value that is not matching any of the levels is *White* (unspecified).

After giving your watch a unique name select the metric that should be watched. Only single metrics are supported.

Select the appropriate unit for the selected metric. Depending on the chosen unit thresholds can be given in values of this unit, e.g. when selecting *Milliseconds* the threshold can be given as `1s` for 1 second.

NOTE	There is limited knowledge what the raw number of the metrics means as not all sources do include this information.

When the name, watched metric and unit attributes are set, check any of the states a

condition should be formulated for and enter the comparison operator, the threshold and the scope.

The below table describes possible choices for each step of a watch condition.

Table 175. Watch Condition Segments

Segment	Type	Description
`is` *operator*	Start	Select the comparison operator to use when comparing the actual value(s) to the threshold. The comparison is of form *actual operator threshold*.
`is` *threshold*	Start	The numeric threshold. The upper/lower limit. With the right unit selected the number can be given with a unit such as `5min` for time units, or `4KB` for bytes.
`is` *scope*	Start	Select a scope for the actual values that are compared to the threshold. The semantics of different scopes are described in a separate table below.
`is` *scope-length*	Start	The length defines how many of the most recent data points of the watched metric are included in the scope of the comparison.
`until` *operator*	Stop	same as for `start`
`until` *scope*	Stop	same as for `start`
`until` *scope-length*	Stop	same as for `start`

The below table describes the semantics of the different types of comparison scopes.

Table 176. Watch Condition Scopes

Scope	Description
`for last`	The comparison must be true for each point in the scope for the condition to be met. Similar to an *all* logic.

Scope	Description
`for average of last`	The comparison must be true for the single average value of all points in the scope for the condition to be met. The number of points included is specified afterward as number of duration from which to consider points.
`in last`	The comparison must be true for at least one of the points included in the scope for the condition to be met. Similar to an *any* logic. The number of points included is specified afterward as number of duration from which to consider points.
`in sample`	The comparison must be true for at least one of the points available for the condition to be met. For this scope type no length is given as all available points are considered. This can vary but usually consists of the last minute of data.

After composing the watch conditions press *Save* to install the watch.

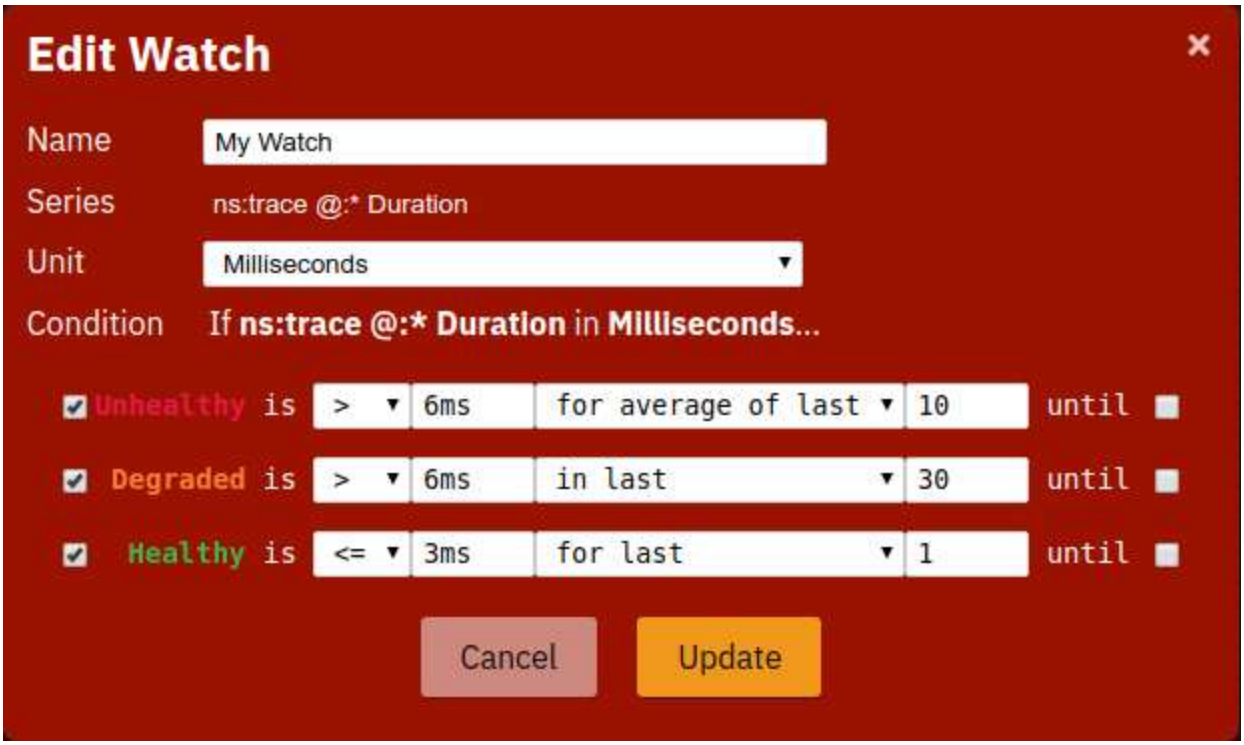
WARNING | If a watch of the same name already exists the existing watch is replaced.

System watches that originate from server modules cannot be changed. But they can be disabled and copied to effectively make them editable. Find more in the edit section below.

WARNING | Watches operate on the DAS server. Their configuration is common for all users. Any change will affect other users.

Editing a Watch

To edit a user watch open the watch manager, scroll down to the list of *User Watches* and click the *Edit* button of the watch that should be edited.



Confirm changes by clicking *Update*.

When changing the name of the watch a new watch is created with the new name and the old watch continues to exist. It can be deleted afterward using the *Delete* button in the *User Watches* list.

WARNING	When changing the name of a watch to a name of an already existing watch this overrides that watch. In other words watches are always identified by their name.

NOTE	System watches cannot be edited. To achieve the effect of a changed system watch disable and copy it. The copy is a user watch that can be adjusted freely. The suggested name *Copy of {original name}* can be changed as long as it is unique.

Enabling or Disabling a Watch

Open the watch manager as an *Administrator* and click the *Disable* or *Enable* button of the watch that should be disabled or enabled.

The change has immediate effect and can easily be "undone" by toggling the state again. If there were any ongoing alerts due to a watch that gets disabled, these alerts stops.

Enabling such a watch again then requires matching the watches start condition again.

Preconfigured Pages

Payara InSight is shipped with a dozen of preconfigured pages. These are in no way privileged over the user-created pages. It is simply a configuration identical to one user can create manually that is hard coded as a preset.

Users can modify these pages freely, but they cannot be deleted.

Core Page

The *Core* page gives an overview of some of the important instance metrics. Each instance in the grid is shown separately.

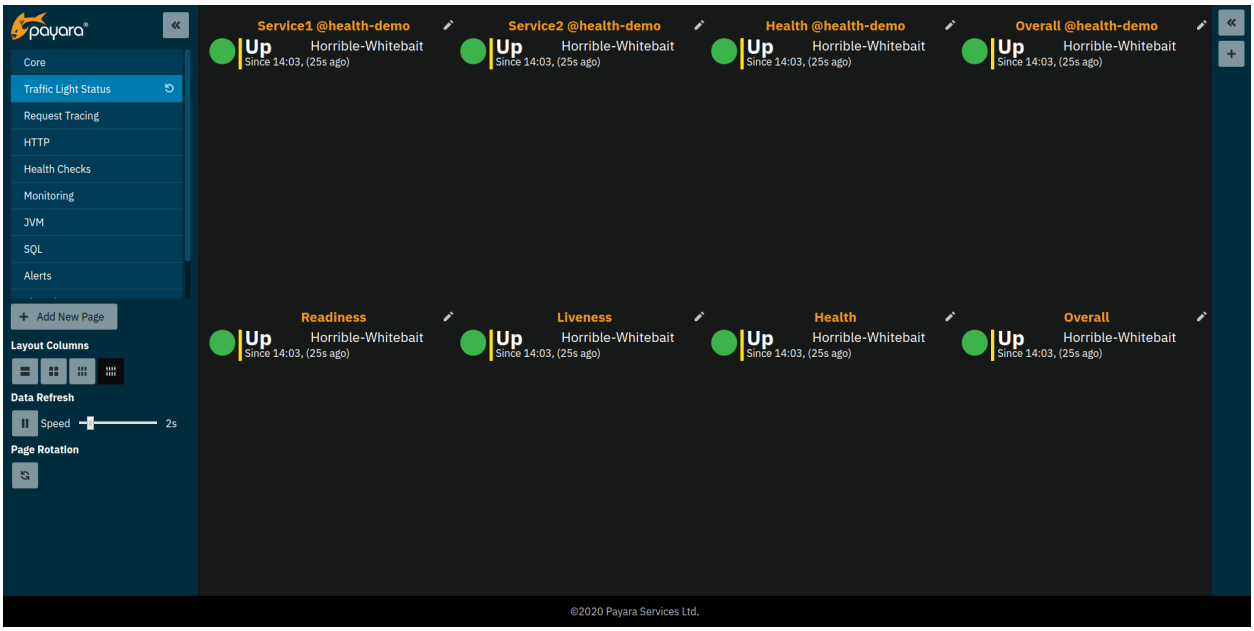
- *Active Sessions*: Total number of active session for the instance.
- *Thread Count*: Total number of threads used by the instance.
- *Request Count*: Number of HTTP request per second processed by the instance.
- *Heap Usage*: Percentage of the current instance maximum heap memory already

used.

- *CPU Usage*: Percentage CPU usage by the instance (not the machine but the process).
- *Thread Pool Current Thread Usage*: Percentage of HTTP thread pool threads currently in use by the instance.

Traffic Light Status Page

The *Traffic Light Status* page (a.k.a. RAG Status for Red-Amber-Green Status) indicates the health status of the server and deployed application using traffic light signals to visualise the current state.



The content of the page is based on Payara Health Checks (needs to be enabled in the server configuration) and MicroProfile Heath checks of deployed applications.

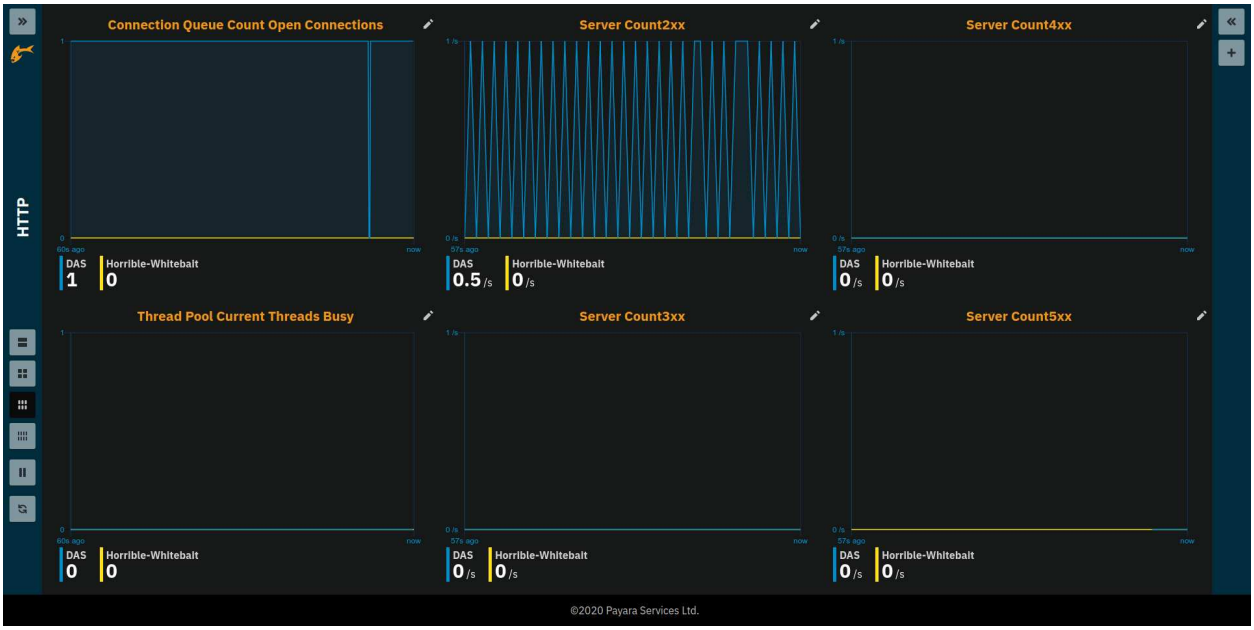
In contrast to most pages this page is not static based on a manually created configuration. Instead, the page is populated with widgets based on a query.

For more details on query pages see sections Adding Pages and Page Settings.

HTTP Page

The *HTTP* page gives a rough overview of the state of the HTTP services of each

instance.



- *Connection Queue Count Open Connections*: Total number of open connection in the connection queue
- *Thread Pool Current Threads Busy*: Number of threads in the HTTP thread pool currently processing an HTTP request.
- *Server Count2xx*: Requests per seconds and instance responding with Success.
- *Server Count3xx*: Requests per seconds and instance responding with Redirection.
- *Server Count4xx*: Requests per seconds and instance responding with Client Error.
- *Server Count5xx*: Requests per seconds and instance responding with Server Error.

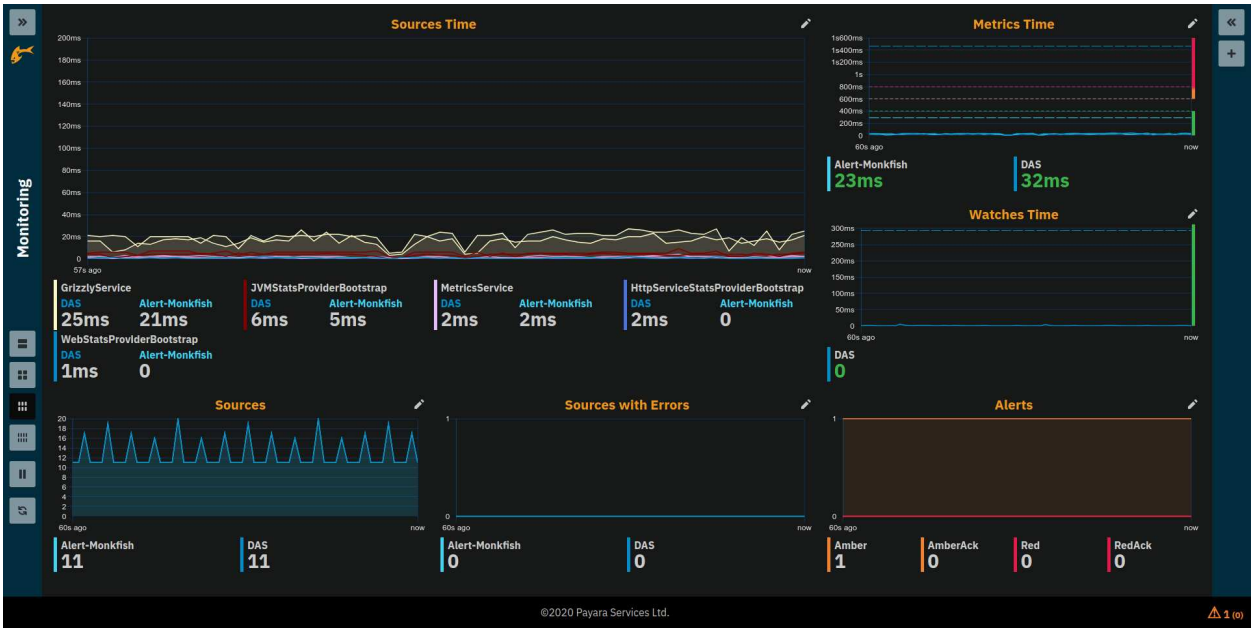
JVM Page

The *JVM* pages gives a general overview of the state of the JVM running Payara Server.

- *Used Heap*: Total size of the used heap memory
- *Heap Size*: Size of the committed (reserved) heap memory (OS level)
- *Loaded Classes*: Total number of loaded classes
- *Unloaded Classes*: Total number of unloaded classes
- *Live Threads*: Total number of threads used by the JVM (including daemon threads)
- *Daemon Threads*: Total number of daemon threads used by the JVM

Monitoring Page

This is mostly useful to identify problems with the monitoring system or to cross-check that unexpected metric values aren't caused by general problems with monitoring.



- *Source Times*: Shows the duration it took to collect metric data from the individual sources within the server. Durations should be well below a second. Usually in two digit millisecond range.
- *Sources*: Number of sources metrics are collected from in a particular second. This varies as some sources are collected in an interval of several seconds.
- *Sources with Errors*: Shows how many of the sources that were collected caused an error during collection that potentially ends the collection of that source so that some metrics usually provided by that source might no longer be collected. This can explain "missing" metrics.
- *Metrics Time*: The total time it took to collect all sources. As collection happens once every second this should be well below 1 second.

  This metric is watched and alerts are created should the time exceed 600ms for multiple times in a row.

  This time can vary a lot depending on the server configurations as sources are enabled and disabled. It can also be influenced strongly by the deployed applications as they can contribute metrics in large numbers.

- *Watches Time*: The total time it took to collect and evaluate all watches. This time is

less critical, but it should usually be well below 1 second.

Typically, in the range of two digit milliseconds.

- *Alerts*: The total number of ongoing (active) alerts.

Health Checks Page

A page that gives an overview of the state of the server health checks. The individual checks need to be enabled in the server's *HealthCheck* configuration.

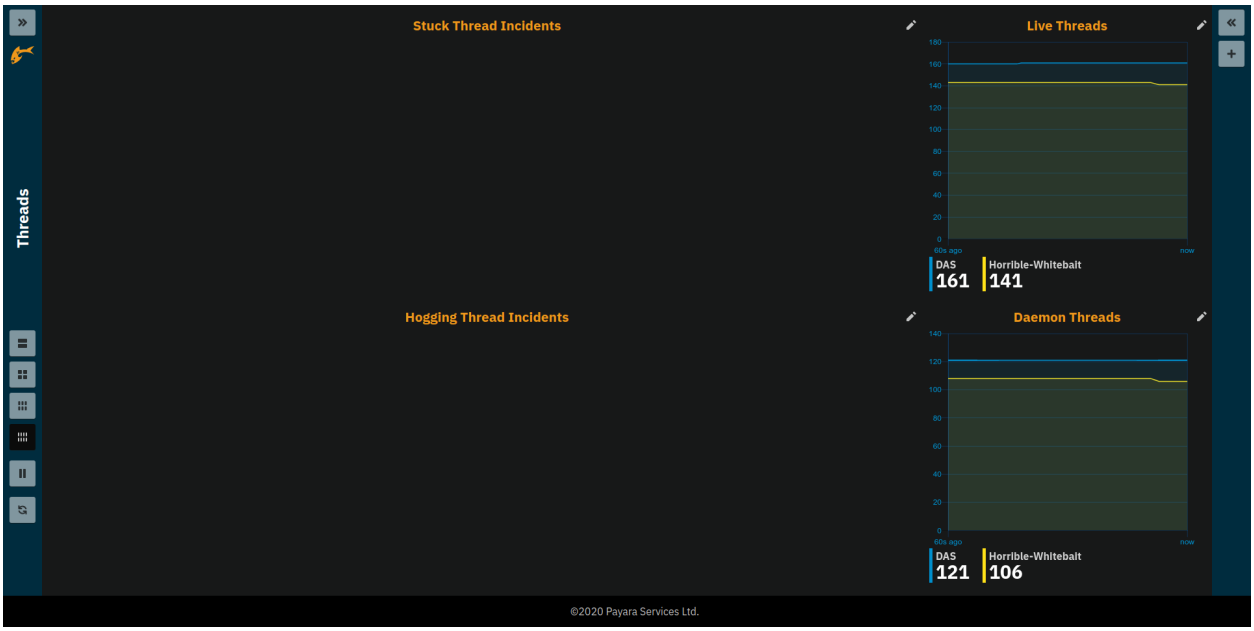
NOTE	It is not required to enable the *HealthCheck service* in the *General* tab in case the checks should only be enabled for metrics based monitoring without causing health check notifier messages.

- *CPU*: CPU Health Check, average percentage of CPU used by the server instance during last 4 seconds
- *GC*: Garbage Collection Health Check, percentage of time used to collect garbage objects during the last 4 seconds
- *Memory*: Machine Memory Usage Health Check, percentage of the total available physical memory used

- *Heap*: Heap Memory Usage Health Check, percent of total available heap memory used by the JVM
- *MP Health*: MicroProfile Health Check, percent of instances that respond with `UP` status to a liveliness health endpoint request (this includes the DAS that is the only instance that is performing this check every 12 seconds)
- *Connection Pool*: Connection Pool Health Check, percent of connections currently used for each JDBC connection pool
- *Alerts*: A list of all health check related alerts

Thread Health Page

The *Threads* page gives detailed information on thread health of the cluster and lists information on stuck or hogging threads. The total number of live and daemon threads are given for reference.



- *Stuck Thread Incidents*: Table listing the most recent information for threads that are considered stuck.

  This is the case if the time passed since a thread started a not yet finished unit of work becomes longer than the set threshold in the *Stuck Threads* health check configuration.

This could be because it is waiting at a lock or the operation takes a considerable amount of time.

- *Hogging Thread Incidents*: Table listing the most recent information for threads that are considered hogging.

  This is the case if the thread's use of the CPU in percent points is higher than the threshold value configured in the *Hogging Threads* health check configuration for `n` times in a row, where `n is 1 + the configured number of retries`.

- *Live Threads*: Total number of threads used by the JVM (including daemon threads)
- *Daemon Threads*: Total number of daemon threads used by the JVM

WARNING	Be aware that the number shown in the example image might not be characteristic. For demonstration purposes the thresholds might be set uncharacteristically low to be able to "force" entries. Usually the threshold for stuck threads should be in region of some seconds to a few minutes. The threshold for hogging threads should be well above 90% with a few retries.

Request Tracing Page

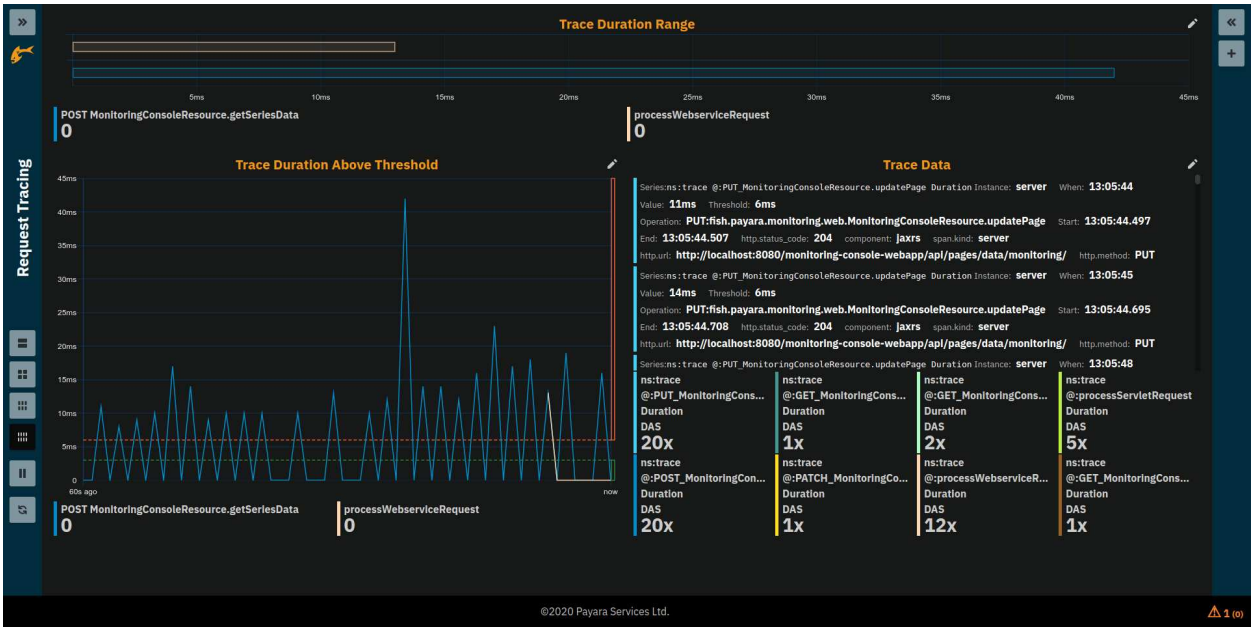
The *Request Tracing* page overview page shows traces that exceeded the set threshold configures as part of the request tracing server configuration.

NOTE	If this page does not show any data, no request has exceeded the threshold or the request tracing has not been enabled. Check server config *Configuration* ▯ *Request Tracing*.

The *Trace Duration Range* widget shows all traces for the same method as a single bar. The length and position of the bar gives the range of total trace duration for all the occurrences that exceeded the threshold.

The bar starts at the minimum duration observed and ends at the maximum duration observed in the last minute. The legend shows the average duration and the method

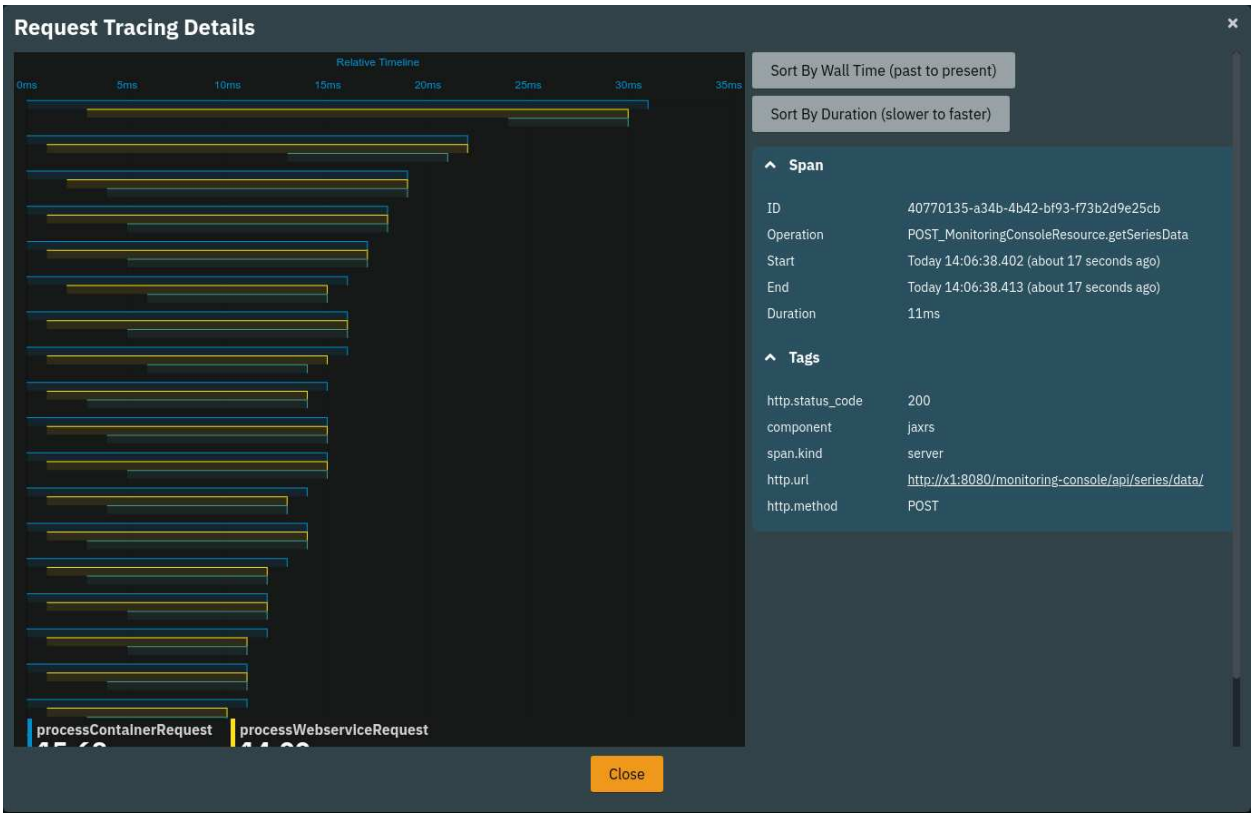
traced (innermost trace span operation)



When checking the minimum and/or maximum checkbox in the widgets *Extras* settings the bar starts and/or ends at these "all time" values instead of the minimum and/or maximum within the last minute.

The *Trace Duration Above Threshold* widget shows a graph where each point marks the duration of a "slow" request at the moment the requests occurred. Possible alerts are shown on top of the graph.

By clicking on one of the bars in the overview's *Trace Duration Range* widget the details of all kept traces for this method can be viewed.



This data corresponds to the traces stored in the trace-store. For each trace, each of the spans corresponds to a bar in this graph.

The details can be sorted by wall-time or by total duration. By default, the traces are sorted longest (slowest) to shortest (fastest) total duration.

On the time axis the spans are shown relative to the start of the trace.

The legend gives average durations for each of the spans occurring for the traced method.

NOTE	The detail view shows a snapshot of the data found in the trace store. Changing the sorting will reload data and reflect an updated snapshot.

When clicking on the bars in the details view the right side panel shows the trace span details of the clicked trace span.

WARNING	Be aware that the graphs in the example images above show unrealistically low numbers due to the way that request tracing was

> configured. In a more realistic configuration, the durations are more likely to be in range of a few hundred milliseconds to some seconds.

SQL Page

The *SQL* page gives insight into slow SQL queries for all connection pools configured in Payara Server.

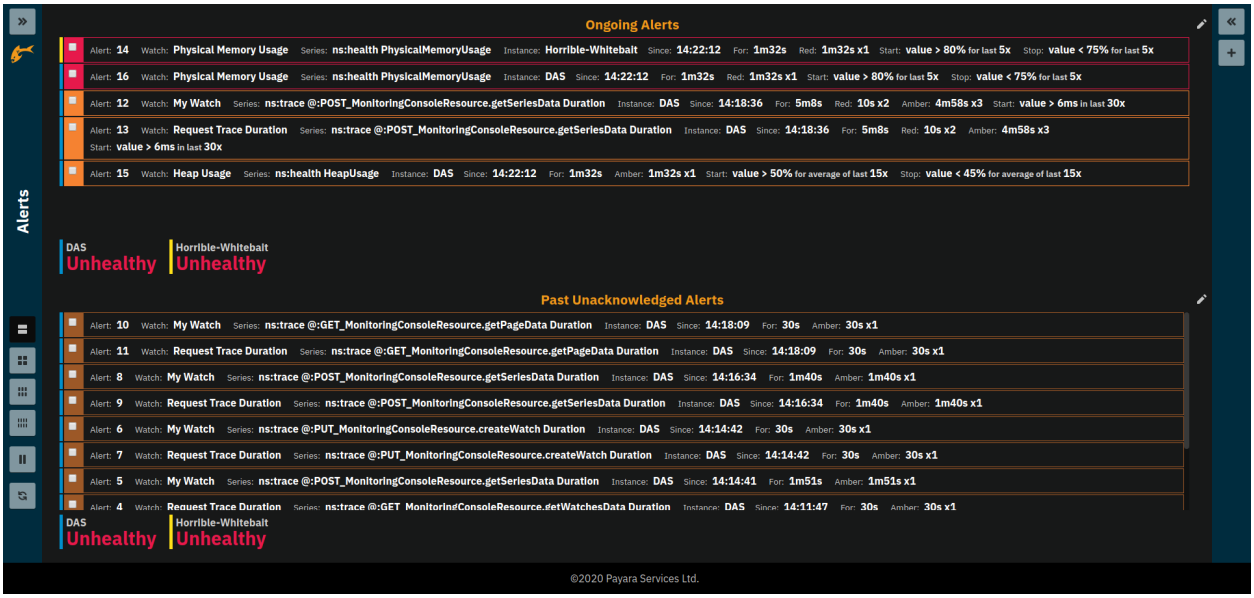
NOTE To capture slow SQL queries the *JDBC Connection Pool* configuration's *Slow Query Log Threshold* in the *Advanced* tab needs to be set to a positive number.

- *Slow SQL Queries*: Shows SQL details for queries that were too slow. *Value* refers to the duration of the SQL query.
- *Worst SQL Execution Time*: Shows SQL query activity. Each data point gives the duration of the slowest query run during last second. Zero means no SQL query was run.
- *Slow SQL Alerts*: A list of alerts caused by too slow SQL queries in reference to the *Slow Query Log Threshold* given in the pool configuration. Single outliers cause a degraded state (amber alerts), outlier groups cause unhealthy state (red alerts).

Alerts Page

The main use of the *Alerts* page is to check for any server alerts. The page is split into two lists: the upper list containing all ongoing alerts, and the lower list containing past alerts.

As always for alert lists both lists are sorted starting with the most recent severe alert progressing to the oldest least severe alert. Annotations associated with alerts are hidden in this view to make room for more alert entries.



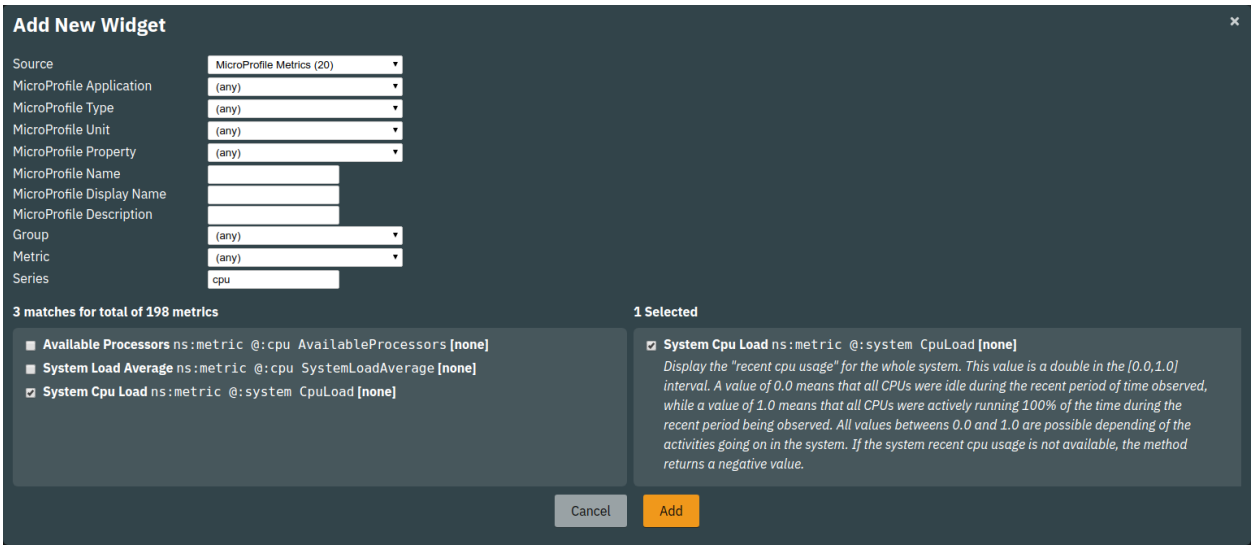
NOTE	Past alerts can be managed by acknowledging the alerts which makes them disappear from the list.

Application Metrics Page

Payara InSight is build with the intention to make visualising user defined metrics really easy. This includes MicroProfile Metrics and MicroProfile Health checks that as defined by deployed application and server itself.

It does not require and further configuration to make them available as metrics within Payara InSight.

In addition, the metrics selection wizard has additional filters specifically filtering on the properties of MicroProfile Metrics.



When MicroProfile Metrics get translated to InSight series the MP tags become the `@` group of the shown series key, the MP *metric* stays the metric name.

The *Application Metrics* page is a page that is automatically populated will MicroProfile Metrics based on a query. This means the metrics shown on the page vary depending on what metrics are defined in applications and the server as MicroProfile metrics.

When populating the page the widgets are configured automatically based on the information available in MicroProfile.

For more details on query pages see Adding Pages and Page Settings.

Experimental Features

This section includes features that aren't fully complete or integrated or which might change drastically or even be removed in future releases.

Pattern Data Series

The term *series* is used for the name or key for a particular metric. Each metric has a unique key or series name.

Usually these names use *tags* for form *key:value* to categorise metrics into groups. For example:

```
ns:jvm MetricName
ns:jvm @:groupName MetricName
```

The series names shown above exemplify conventions used throughout Payara InSight. The first grouping level usually is the `ns` (short for namespace), the second the `@` (short for *group*).

In the example the metrics belong to the `jvm` namespace and the `groupName` group within that namespace.

When metrics are selected using the wizard only fully qualified metric series can be selected. This is a limitation of the selection wizard. On the technical level a series can be either a fully qualified series name or a series pattern.

Patterns contain wild-cards to match not only one specific metric but a set of metrics. Wild-cards can occur in different places. Examples of this can be found in the query page's *Query Series* setting.

For example the *Traffic Light Status* page uses the query `*boldns:health ?:* *`.

A `*bold*` wild-card for the metric name means *any metric name*.

The `*bold?:*` combines a wild-card (tag) value (`*bold*`) with a wild-card tag name (

`?`). This means *any tag name* (including none and multiple of them) with *any value*.

So the full `*boldns:health ?:* *` matches any series that is in `ns:health`, weather or not is has more tags, while `*boldns:health *` only would match series in `ns:health` which do not have further tags.

NOTE	This feature can be used to automatically show all metrics of a certain pattern in a single widget as it is done by the request tracing page.

Diagnostics and Troubleshooting

Diagnostics Tool

Payara Server has a diagnostic tool that on demand collects the following information from the server runtime.

- `domain.xml` configuration file
- Server logs
- Notification logs
- Access logs
- A JVM thread dump.
- A JVM Heap Dump
- JVM Report

The collection of this data can be targeted towards a specific remote or local server instance, deployment group or cluster. By default, everything is collected from the DAS, and each instance, deployment group and cluster configured in the domain.

If the target(s) is(are) offline, only the `domain.xml` configuration file, server, access and notification logs will be collected.

NOTE	If you are not using the default *domain1*, you must specify the domain to collect diagnostics from.

TIP

The diagnostics tool can be manually upgraded to a newer version in case it is needed. To do so, download the latest 2.x JAR release from Nexus using this link and then replace the older version into your Payara Server installation under the `as-install/glassfish/lib/asadmin` folder.

The tool is also updated at every release of Azul Payara.

Usage

The diagnostic tool can be used through the collect-diagnostics Asadmin CLI subcommand, which will collect the relevant data and store it locally in a ZIP bundle that can be used for troubleshooting purposes when raising support tickets on the support portal.

Using this command will allow both customers and support staff to work faster on the investigation and analysis of production and development incidents.

TIP

The `--obfuscate` parameter will hide sensitive data in the `domain.xml` such as passwords, addresses and hostnames.
Use it if you're afraid of sensitive data being shared along with the domain configuration.

NOTE

The data collected is stored in a ZIP file in your user home directory, the output directory by default.

The resulting compressed ZIP file will have a directory structure similar to the following:

```
deploymentGroup1
¬¬¬ instance1
¬   ¬¬¬ instance1-domain.xml
¬   ¬¬¬ instance1-jvm-report.txt
¬   ¬¬¬ instance1-server.log
¬   ¬¬¬ instance1-notification.log
¬   ¬¬¬ instance1-server_access_log.txt
¬   ¬¬¬ instance1-thread-dump.txt
¬¬¬ instance2
    ¬¬¬ instance2-domain.xml
    ¬¬¬ instance2-jvm-report.txt
    ¬¬¬ instance2-server.log
    ¬¬¬ instance2-notification.log
```

```
    ¬¬¬ instance2-server_access_log.txt
    ¬¬¬ instance2-thread-dump.txt
```

Where the data is grouped in sub-folders segregated in all targets (deployment groups, clusters and instances) scanned by the command.

Example

The following example shows how to collect the data of all instances that belong to the `deploymentGroup1` deployment group in the `mydomain` domain:

```
asadmin> collect-diagnostics --target deploymentGroup1 mydomain

Directory selected /home/user/payara/diagnostics/mydomain-2024-04-15T11-42-36Z
Collecting domain.xml from instance2
Attempting to create missing path at /home/user/payara/diagnostics/mydomain-2024-
04-15T11-42-36Z/deploymentGroup1/instance1
Obfuscating /home/user/payara/diagnostics/mydomain-2024-04-15T11-42-
36Z/deploymentGroup1/instance1
Successfully obfuscated /home/user/payara/diagnostics/mydomain-2024-04-15T11-42-
36Z/deploymentGroup1/instance1
Collecting logs from instance2
Collecting jvm report from instance2
Collecting thread dump from instance2
Collecting Heap Dump from instance2
This version of Payara does not support heap dump generation.
Collecting domain.xml from instance1
Attempting to create missing path at /home/user/payara/diagnostics/mydomain-2024-
04-15T11-42-36Z/deploymentGroup1/instance1
Obfuscating /home/user/payara/diagnostics/mydomain-2024-04-15T11-42-
36Z/deploymentGroup1/instance1
Successfully obfuscated /home/user/payara/diagnostics/mydomain-2024-04-15T11-42-
36Z/deploymentGroup1/instance1
Collecting logs from instance1
Collecting jvm report from instance1
Collecting thread dump from instance1
Collecting Heap Dump from instance1

Command collect-diagnostics executed successfully.
```

Collect Diagnostics with an Unavailable DAS

Diagnostics can be collected directly from a running instance in case its corresponding DAS it's not available by specifying the admin port of the target instance (see options), like this:

```
asadmin> -p 24848 collect-diagnostics --target deploymentGroup1 mydomain
```

NOTE	This command must be run locally on the host of the node where this instance is located.

Upgrade Guide

Upgrading Payara Server

There are three valid methods of fully upgrading to a newer release of Payara Enterprise. Any of the following methods would work in most circumstances:

NOTE	If you are migrating from Azul Payara Community to Azul Payara, you must use the backup and restore method.

1. Backing up and restoring the existing configuration to a new installation. "Read Backup and Restore Method"
2. Maintaining completely separate domain and node directories and pointing the new version to the existing directories. "Read Domain and Node Directories Method"
3. Using the Payara Upgrade Tool. "Read Payara Upgrade Tool Method"

JDK 11: Upgrade Considerations

IMPORTANT	Applies when upgrading to Payara Server 5.192 and superior releases

The `domain.xml` configuration file contains instructions for handling JDK 11. Without these instructions, Payara Server will not start up on JDK 11 and/or will throw many exceptions during runtime.

If older, pre-5.192, domains are being migrated this has to be taken into account. The following `jvm-options` have to be added to the `java-config` element in `domain.xml`:

```
<jvm-options>[9|]--add-opens=java.base/jdk.internal.loader=ALL-UNNAMED</jvm-
options>
<jvm-options>[9|]--add-opens=jdk.management/com.sun.management.internal=ALL-
UNNAMED</jvm-options>
<jvm-options>[9|]--add-exports=java.base/jdk.internal.ref=ALL-UNNAMED</jvm-options>
<jvm-options>[9|]--add-opens=java.base/java.lang=ALL-UNNAMED</jvm-options>
```

```
<jvm-options>[9|]--add-opens=java.base/java.nio=ALL-UNNAMED</jvm-options>
<jvm-options>[9|]--add-opens=java.base/sun.nio.ch=ALL-UNNAMED</jvm-options>
<jvm-options>[9|]--add-opens=java.management/sun.management=ALL-UNNAMED</jvm-
options>
<jvm-options>[9|]--add-opens=java.base/sun.net.www.protocol.jrt=ALL-UNNAMED</jvm-
options>
<jvm-options>[9|]-Xbootclasspath/a:${com.sun.aas.installRoot}/lib/grizzly-npn-
api.jar</jvm-options>
```

The following `jvm-options` have to be removed:

```
<jvm-options>
-Djava.endorsed.dirs=${com.sun.aas.installRoot}/modules/endorsed${path.separator}${
com.sun.aas.installRoot}/lib/endorsed</jvm-options>
<jvm-options>
-Djava.ext.dirs=${com.sun.aas.javaRoot}/lib/ext${path.separator}${com.sun.aas.javaR
oot}/jre/lib/ext${path.separator}${com.sun.aas.instanceRoot}/lib/ext</jvm-options>
```

JDK 17 and 21: Upgrade Considerations

IMPORTANT	Applies when upgrading to Payara Enterprise 5.39.0 and superior releases

The `domain.xml` configuration file contains instructions for handling JDKs 17 and 21. Without these instructions, Payara Server will not start up on JDK 17 and/or will throw many exceptions during runtime.

If older, pre-5.192, domains are being upgraded follow the JDK 11 upgrade considerations steps first, then come back to the JDK 17 & 21 upgrade considerations.

If older, pre-5.39.0, domains are being upgraded to Payara Server 5.39.0 or above, the following `jvm-options` have to be added to the `java-config` element in `domain.xml`:

```
<jvm-options>[17|]--add-exports=java.base/sun.net.www=ALL-UNNAMED</jvm-options>
<jvm-options>[17|]--add-exports=java.base/sun.security.util=ALL-UNNAMED</jvm-
options>
<jvm-options>[17|]--add-opens=java.base/java.lang.invoke=ALL-UNNAMED</jvm-options>
<jvm-options>[17|]--add-opens=java.desktop/java.beans=ALL-UNNAMED</jvm-options>
<jvm-options>[17|]--add-opens=java.base/java.io=ALL-UNNAMED</jvm-options>
<jvm-options>[17|]--add-exports=jdk.naming.dns/com.sun.jndi.dns=ALL-UNNAMED</jvm-
options>
<jvm-options>[17|]--add-exports=java.naming/com.sun.jndi.ldap=ALL-UNNAMED</jvm-
options>
```

Payara Server 5 to Payara Server 6 Migration Considerations

Package names for Corba classes have changed from `com.sun` to `org.glassfish`, therefore if you use the `java.naming.factory.initial` system property you should set it to `org.glassfish.jndi.cosnaming.CNCtxFactory` for Payara Server 6.

Payara Server Upgrade Tool

This tool can be used to upgrade an existing Payara Server installation to a newer version. The Upgrade Tool is bundled in Payara Server by default and is maintained as a separate JAR artifact under the `fish.payara.extras:payara-upgrade-tool` GAV coordinates.

TIP	The upgrade tool can be manually upgraded to a newer version in case it is needed. To do so, download the latest 2.x JAR release from Nexus using this link and then replace the older version into your Payara Server installation under the `${PAYARA_SERVER_INSTALL}/glassfish/lib/asadmin` folder.

Upgrading to Payara Server 7

It is possible to upgrade a Payara Server `6.x` domain to version `7.x`. The process should not be any different as to upgrading between different Payara 6 minor versions, though with the following caveats:

1. When performing an upgrade from Payara 6 to Payara 7, make sure that you're using JDK 21 as this is the minimum JDK version required to run Payara Server 7.

IMPORTANT	It is recommended that users should run the latest version of Payara Server 6.x before attempting to upgrade to version 7.x.

Asadmin CLI Commands

The following are the Asadmin CLI commands available to the upgrade tool.

Upgrade Server Command

Usage: `asadmin> upgrade-server`

Aim: This command can be used to upgrade Payara Server to the specified distribution along with all SSH nodes in the `domain.xml` configuration file.

The command will also back up the old version and domains in the default location (`payara7/glassfish/domains`) or the location specified using the `--domaindir` option to allow for rolling back.

Command Options

Option	Type	Description	Default	Mandatory
`--username`	String	The username for the Payara Nexus server.		Yes, unless `--usedownloaded` is specified
`--version`	String	Specifies the version number of the new version of Payara to use.		Yes, unless `--usedownloaded` is specified
`--distribution`	Enum	Specifies the distribution of Payara Server to download. This can payara, payara-web, payara-ml, or payara-web-ml.	payara	No
`--stage`	Boolean	Determines if the upgrade should be installed in-place, or staged into "*x.new*" directories. Scripts are provided that can be used to apply and rollback the staged install. This option is automatically enabled when running the command on Windows.	False on Unix, True on Windows	No

Option	Type	Description	Default	Mandatory
`--domaindir`	String	The directory containing the domains. The domains in this directory are backed up, and their config is used to determine the nodes which will also be upgraded.	`${as-install}/domains`	No
`--usedownloaded`	String	The path to the local Payara Server zip archive to use for upgrading instead of downloading from the Payara Enterprise repository. When this parameter is specified, the `--username`, `--password`, and `--version` parameters are not required.		No
`--nodes`	String	Specifies the nodes to upgrade if you want to selectively upgrade certain nodes. Provide as a comma-separated list of node names.	All nodes in the cluster	No

Example

This example upgrades a Payara Web distribution to version 7.1.0

```
asadmin> upgrade-server --username example-user --distribution web --version 7.1.0
```

To selectively upgrade specific nodes, use the `--nodes` option with a comma-separated list of node names. This example upgrades the domain administration server and one remote node, leaving other nodes to be upgraded later:

```
asadmin> upgrade-server --username example-user --version 7.1.0 --nodes localhost-domain1,remotehost1-node1
```

Rollback Server Command

Usage: `asadmin> rollback-server`

Aim: This command can be used to rollback Payara Server to the point before the upgrade-server command was run, restoring the most recent backup of the domain (expected to be the backup created during execution of the `upgrade-server`

command).

WARNING	This command is not supported on Windows OS, please use the `rollbackUpgrade.bat` script instead.

Command Options

Option	Type	Description	Default	Mandatory
`--domain dir`	String	The directory containing the domains. The domains in this directory are backed up, and their config is used to determine the nodes which will also be upgraded.	`${as-install}/domains`	No
`--nodes`	String	Specifies the nodes to rollback if you want to selectively rollback certain nodes. Provide as a comma-separated list of node names.	All nodes in the cluster	No

Staged Upgrades

When the `upgrade-server` command is either used on Windows or with the `--stage` option enabled, the new server files are installed next to the current installation in various *.new* directories (e.g. `payara7/glassfish/bin.new`). The following helper scripts are available to interact with staged upgrades.

Apply Staged Upgrade Script

Usage: `> ./payara7/glassfish/bin/applyStagedUpgrade`

Aim: This script is used to apply an upgrade staged using the `upgrade-server` command. It will move the current installation into *.old* directories, and the staged *.new* installation into the expected "current" location. It will then upgrade the nodes of the domains in the default domain dir, or the domains in the directory provided using `--domaindir`

Command Options

Option	Type	Description	Default	Mandatory
`--domaindir`	String	The directory containing the domains. The config of the domains in this directory are used to determine the nodes which will also be upgraded.	${as-install}/domains	No
`--nodes`	String	Specifies the nodes to upgrade if you want to selectively upgrade certain nodes. Provide as a comma-separated list of node names.	All nodes in the cluster	No

Rollback Upgrade Script

Usage: `> ./payara7/glassfish/bin/rollbackUpgrade`

Aim: This script is used to rollback a server upgrade applied using the `applyStagedUpgrade` script. It will move the *.old* installation back into the expected "current" location, and the applied upgrade back into *.new* directories. It will then rollback the nodes of the domains in the default domain dir, or the domains in the directory provided using `--domaindir`

Command Options

Option	Type	Description	Default	Mandatory
`--domaindir`	String	The directory containing the domains. The config of the domains in this directory are used to determine the nodes which will also be rolled back.	`${as-install}/domains`	No
`--nodes`	String	Specifies the nodes to rollback if you want to selectively rollback certain nodes. Provide as a comma-separated list of node names.	All nodes in the cluster	No

Cleanup Upgrade Script

Usage: `> ./payara7/glassfish/bin/cleanupUpgrade`

Aim: This script is used to clean up any leftovers from a staged upgrade: any *.old* folders and any *.new* folders will be deleted.

WARNING	Use of this script will prevent you from rolling back or applying a

staged upgrade.

Configure Logging Levels

The upgrade tool commands and helper scripts will print a set of minimum details of the operations executed (upgrade, staging, rollback). For troubleshooting scenarios, or if wanting to review in detail all executed actions, the following 2 environment variables are available to control the level of logging done by the Upgrade tool:

`AS_DEBUG`: Set to `true` to configure the Upgrade Tool's logging level to `FINER`.
`AS_TRACE`: Set to `true` to configure the Upgrade Tool's logging level to `FINESET`.

These variables can also be configured as system properties in the Asadmin CLI script file located in the `{as-install}/bin` folder like this:

```
AS_INSTALL=`dirname "$0"`/../glassfish
AS_INSTALL_LIB="$AS_INSTALL/lib"
. "${AS_INSTALL}/config/asenv.conf"
JAVA=java
#Depends upon Java from ../config/asenv.conf
if [ ${AS_JAVA} ]; then
    JAVA=${AS_JAVA}/bin/java
fi

exec "$JAVA" -DAS_DEBUG=true  -XX:+IgnoreUnrecognizedVMOptions -jar
"$AS_INSTALL_LIB/client/appserver-cli.jar" "$@"
```

IMPORTANT Remember to turn off these logger level settings after executing a server upgrade, as this setting will affect all future executions of any Asadmin CLI commands, which will cause them to print out more information than usual.

Zero Downtime Upgrade

IMPORTANT This feature requires payara-upgrade-tool.jar from Payara v5.83.0 or v6.34.0 and above and assumes you are running a minimum of two SSH nodes.

By default, upgrading Payara requires that the domain and all SSH nodes are stopped. This feature allows the upgrading of individual SSH nodes, eliminating downtime. Only

upgrades in the same major version are supported 6.x to 6.x, 7.x to 7.x. Zero Downtime upgrades between 6.x and 7.x are not supported however it is possible to achieve if you can avoid restarting any of the 6.x nodes while running 7.x DAS.

Steps to Upgrade

stop the DAS

```
asadmin> stop-domain
```

upgrade the domain

```
asadmin> upgrade-server --username example-user --distribution web --version 7.1.0
--nodes localhost-domain
```

Stop one SSH node to upgrade

```
asadmin> stop-node-ssh node1
```

Upgrade one SSH node

```
asadmin> upgrade-server --username example-user --distribution web --version 7.1.0
--nodes node1
```

Start the upgraded SSH node

```
asadmin> start-node-ssh node1
```

Repeat the last 3 steps with the remaining SSH nodes to upgrade. Start the DAS

```
asadmin> start-domain
```

TIP	Use the `_list-node-versions` subcommand to help keep track of the current version of each node, including staged and rollback versions.
WARNING	the upgrade can cause issues if the nodes are forming a cluster with Hazelcast. When 2 versions of Payara use different versions of

> Hazelcast, the upgraded node will fail to start as long as nodes using the older versions are running. If the upgraded node/DAS fails to start, you may have to upgrade everything else before restarting the nodes and the DAS. Alternatively, you can avoid these issues by disabling the data grid for every nodes. Below are the known combinations causing issues:

Known Upgrade Issues

Existing	Upgrade	Result
7.0.0	7.2.0	No issues

Backup and Restore Upgrade Method

You will need to download the version of Payara Enterprise you are planning to upgrade to before beginning the backup and restore method of upgrading Payara.

Backup a Payara Server Domain

NOTE The `backup-domain` command requires the domain is not running, if this method is chosen for a production system, an arranged period of downtime would be required.

To create a backup of a domain, the asadmin `backup-domain` command is used:

```
asadmin backup-domain {domain_name}
```

The resulting ZIP file will be saved in a destination with this structure:

`{backupDir}/{domain_name}/<domain-name>_<YYYY_MM_DD>_v<backup-number>`

Where the backup number is a consecutive integer representing how many backups of the domain have been made.

You can get additional details of the backup operation by using the `--long` argument when running the command, which will output the following data:

```
asadmin> backup-domain --backupDir /home/james/payarabackup --long domain1

Backed up domain1 at Mon Jul 04 16:25:58 BST 2022.

Description               : domain1 backup created on 2022_07_04 by user james
GlassFish Version         : Payara Server Enterprise  5.39.0 #badassfish (build
529)
Backup User               : james
Backup Date               : Mon Jul 04 16:25:58 BST 2022
Domain Name               : domain1
Backup Type               : full
Backup Config Name        :
Backup Filename (origin)  :
/home/james/payarabackup/domain1/domain1_2022_07_04_v00001.zip
Domain Directory          : /home/james/payara-
5.39.0/payara5/glassfish/domains/domain1

Command backup-domain executed successfully.
```

Read more about the "Backup Domain Command"

Restore a Payara Server Domain

IMPORTANT	Since instances are stored separately from a domain, and they may be on remote hosts, the command will have no effect on restoring instance data.

To restore from a backup, the asadmin `restore-domain` command is used:

```
asadmin restore-domain --filename {path_to_backup} {domain_name}
```

You can get additional details of the restore operation by using the `--long` argument when running the command, which will output the following data:

```
asadmin> restore-domain --filename
/home/james/payarabackup/domain1/domain1_2022_07_04_v00001.zip --long domain1

Restored the domain (domain1) to /home/james/PayaraInstalls/payara-
5.40.0/payara5/glassfish/domains/domain1

Description               : domain1 backup created on 2022_07_04 by user james
GlassFish Version         : Payara Server Enterprise  5.39.0 #badassfish (build
529)
Backup User               : james
Backup Date               : Mon Jul 04 16:25:58 BST 2022
Domain Name               : domain1
Backup Type               : full
Backup Config Name        :
Backup Filename (origin)  :
```

```
/home/james/payarabackup/domain1/domain1_2022_07_04_v00001.zip
Domain Directory          : /home/james/PayaraInstalls/payara-
5.39.0/payara5/glassfish/domains/domain1

Command restore-domain executed successfully.
```

Read more about the "Restore Domain Command"

Restoring an Instance

Since all data in an instance directory is kept in sync with the DAS, there is no added data (besides logs, caches and some MQ data when a file-based store is used) so an instance can be completely restored simply by making sure there is an empty directory with the name of the instance inside a node folder of the correct name.

For example: `payara5/glassfish/nodes/localhost-example/myInstance`

Running the `start-local-instance` command with the `sync` property set to `full` sync, will restore the instance completely:

```
asadmin> start-local-instance --sync=full myInstance
```

Read more about the "Start Local Instance Command"

Considerations for Openmq

If OpenMQ (Payara Server's JMS broker) is being used, the default configuration is to use EMBEDDED mode, meaning that Payara Server will fully manage the JMS broker within the same JVM as the server instance and a file-based store is used for persistent messages.

The alternative option of LOCAL allows for the use of an "enhanced" broker cluster, whereby the JMS broker instances are started as separate JVMs and use a database connection for storage. Assuming the database is highly available, this completely avoids any danger when needing to use --sync=full on the instance.

Domain and Node Directories Upgrade Method

You can also upgrade Payara Enterprise by creating the domain and node directories in a directory that is separate from the installation. For example, if you downloaded *Payara*

Server 5.35.0, created a domain, and then wanted to use *Payara Server 5.40.0*, you would use the following process:

```
/opt/payara/payara-5.35.0/payara5 bin/asadmin create-domain --domaindir /tmp
myTempDomain
/opt/payara/payara-5.35.0/payara5 cd ../../payara-5.40.0/payara5
/opt/payara/payara-5.40.0/payara5 bin/asadmin start-domain --domaindir /tmp
myTempDomain
```

This will use the *5.35.0* installation to create a new domain, and then run it with the *5.40.0* installation.

Nodes can also be created in a separate directory by using the `--nodedir` property when creating a node.

```
/opt/payara/payara-5.35.0/payara5 bin/asadmin create-node --nodedir /tmp
myLocalNode
```

By using this method, the user configuration is always kept separate to the installation, and rollback is as simple as using the previous installation directory.

If you are using Payara Server on **Linux**, you could take advantage of symbolic links. When your domain directory is in a non-standard location, you need to specify its location with each `start`/`stop`/`restart` command. If you were to create a symbolic link to the domain directory inside the default domains directory, then all of your existing scripts will carry on as normal.

This means that any rollback is as simple as stopping the server, updating a symbolic link and starting it again.

If you choose not to use symbolic links, the asadmin `start-domain` command has a `--domaindir` option which allows you to specify the location of a domain directory. So you could have a directory structure like the following:

```
/opt/payara/5.35.0/payara5/...
/opt/payara/5.38.0/payara5/...
/opt/payara/5.40.0/payara5/...
/opt/payara/domains/myDomain
```

```
/opt/payara/nodes/myLocalNode
```

Then you could start your domain with whatever version of Payara Server you wanted, by running the start-domain command from the corresponding Payara installation directory.

```
/opt/payara/5.35.0/payara5/bin/asadmin start-domain --domaindir /opt/payara/domains
myDomain
/opt/payara/5.38.0/payara5/bin/asadmin start-domain --domaindir /opt/payara/domains
myDomain
/opt/payara/5.40.0/payara5/bin/asadmin start-domain --domaindir /opt/payara/domains
myDomain
```

NOTE The `--domaindir` option specifies the parent directory for your domain, and where you would store all of your domains if needed.

Just as you can create nodes by using the `--domaindir` property, you can also start nodes in the same way.

```
/opt/payara/5.40.0/payara5/bin/asadmin start-local-instance --nodedir
/opt/payara/nodes myLocalNode
```

Payara Micro Documentation

Payara Micro Documentation

This section covers how to use Payara Micro.

Payara Micro is an embedded release of Payara built from the Payara Embedded Web release. It allows you to deploy and run WAR files from the command line with a single command, and also features automatic, dynamic clustering with Hazelcast.

Documentation Conventions

1. Any commands listed will be written assuming they are being run from the same directory as the Payara Micro JAR file.
2. Any commands listed will also be written assuming that the system environment variables have been set up to have Java on the system Path.

3. Any paths listed throughout the documentation will use the Unix/Linux file path structure (forward slashes).
4. The owning class of an API method will not be explicitly stated unless it is not clear if an instance has been started or not; methods that operate on Payara Micro instances before they have been bootstrapped (instances that have not yet been started) are contained in the `PayaraMicro` class, whereas methods that operate on bootstrapped instances (running instances) are contained within the `PayaraMicroRuntime` class.

Docker Images

Payara Services Limited provides official Payara Micro Docker Images which are updated every release.

These images are hosted on Docker hub in the following locations:

- Payara Micro Docker Image

> **TIP** Payara Enterprise customers have access to a private Docker repository containing Payara Enterprise Edition equivalent images, which are also released and updated every month.

Maven Support

This section describes Apache Maven support for Payara Micro.

Maven Central Artifacts

Payara Micro is available in Maven Central, allowing you to include it as a dependency in your project's POM. This allows you to easily add the required Payara Micro classes and methods to your application to use Payara Micro programmatically.

To do this, use the following GAV coordinates:

```
<dependency>
    <groupId>fish.payara.extras</groupId>
    <artifactId>payara-micro</artifactId>
    <version>7.2.0</version>
```

```
</dependency>
```

IMPORTANT	The "Public API" dependency must be also added to the project's POM when developing applications that use proprietary features (`@Clustered` or `@NamedCache` for example).

Run Payara Micro Using the Maven Plugin

There is a maven plugin built specifically for use with Payara Micro. It has goals to `bundle` the application into an Uber JAR, `start` and `stop` Payara Micro. This plugin is "fully documented in the Ecosystem section".

Run Payara Micro Using the Netbeans IDE

There is an Azul Payara tools plugin built for the Apache NetBeans IDE, specifically designed for Payara Micro to create and manage application actions (e.g., run, reload, and debug).

This plugin is fully documented in the "Payara Micro Apache NetBeans IDE" integration.

Payara Micro Configuration and Management

Micro Management

Stopping and Starting Instances

Starting an Instance

This section details the very basics of starting a Payara Micro instance.

Starting an Instance from the Command Line

To start an instance of Payara Micro from the command line, you simply run the JAR:

```
java -jar payara-micro.jar
```

This single command is all you need to run Payara Micro instances; additional configuration options are all part of this command.

Starting an Instance from Persistent Configuration Directory

When using "persistent root configuration directory", it is possible to generate a

launcher bound to that directory using the command:

```
java -jar payara-micro.jar --outputlauncher --rootdir $rootDirectory
```

This will create file `$rootDirectory/launch-micro.jar` and exit immediately. After that, it is possible to launch Payara Micro with plain

```
java -jar $rootDirectory/launch-micro.jar
```

This makes configuration directory self-sufficient and the distribution file (`payara-micro.jar`) is no longer needed.

Starting an Instance Programmatically

You need to import two classes from the `fish.payara.micro` package (contained in the Payara Micro JAR, see section "Maven Support" for instructions on importing this JAR with **Maven**):

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicro;
```

You can then start an instance with default settings by calling the `bootstrap()` method from the `PayaraMicro` class. This initialization will throw a `BootstrapException` exception, so you will need to surround it with a **try-catch**, or have the parent method throw the exception.

A simple example is as follows:

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicro;

public class EmbeddedPayara{

    public static void main(String[] args) throws BootstrapException{
        PayaraMicro.bootstrap();
    }
}
```

To start an instance with non-default settings (configuration changes or deploying

applications upon startup), you have to call the `getInstance()` method before using `bootstrap()`. More details on this can be found in the "Configuring an Instance Programmatically" section. The use of the `getInstance()` method does not exclude you from using the default settings however; calling the `bootstrap()` method on its own (as shown in the example above) is functionally equivalent to calling the `bootstrap()` method directly after the `getInstance()` method, such as in the example below:

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicro;

public class EmbeddedPayara{

    public static void main(String[] args) throws BootstrapException{
        PayaraMicro.getInstance().bootStrap();
    }
}
```

The `bootStrap()` method returns a `PayaraMicroRuntime` object instance, which is comes with various methods that afford you control over the instance and its cluster after you have bootstrapped it. To take advantage of this, you have to initialize a `PayaraMicroRuntime` object from the `bootStrap` method, like so:

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicro;
import fish.payara.micro.PayaraMicroRuntime;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
        PayaraMicroRuntime instance = PayaraMicro.bootStrap();
    }
}
```

Stopping an Instance

This section describes how to shut down a Payara Micro instance.

Stopping an Instance from the Command Line

Payara Micro instances can be stopped by either:

- Using `CTRL-C` on the terminal in which Payara Micro is running.

- Sending a kill signal to the process ID of Payara Micro

Stopping an Instance Programmatically

To shut down a Payara Micro instance programmatically, you will need to use the `shutdown()` method of the `PayaraMicro` or `PayaraMicroRuntime` class.

The `shutdown()` method of the `PayaraMicro` class must be called on the instance of Payara Micro that you want to shut down, so will realistically only be used on a `PayaraMicro` instance variable:

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicro;

public class EmbeddedPayara{

    public static void main(String[] args) throws BootstrapException{

        PayaraMicro micro = PayaraMicro.getInstance();
        micro.bootStrap();
        micro.shutdown();
    }
}
```

The `shutdown()` method of the `PayaraMicroRuntime` class is tied to an instance variable, so is used in much the same way:

```
import fish.payara.micro.PayaraMicro;
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicroRuntime;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{

        PayaraMicroRuntime instance = PayaraMicro.bootstrap();
        instance.shutdown();
    }
}
```

Configuring an Instance

This section details how to configure a Payara Micro instance.

Configuring an Instance from the Command Line

As described in "Deploying From the Command Line", the starting and configuration of an instance can be done in its entirety on one line.

The options available can be seen by running the JAR with the `--help` option, or by consulting the "Payara Micro Command Line Options" section.

The general structure of starting, configuring, and deploying an application to an instance is as follows:

```
java -jar payara-micro.jar _--option1_ _--option2_ ...
```

As an example, see below for starting an instance with a non-default HTTP port:

```
java -jar payara-micro.jar --port 2468
```

Separating Configuration from Production Run

An instance can be configured separately, but only when "persistent root configuration directory" is specified by means of command line argument `--rootDir`.

When switch `--warmup` is supplied all configuration and deployment command line parameters are applied to configuration directory and the instance immediately shuts down:

```
java -jar payara-micro.jar --option1 --option2 deployment.war --warmup
```

Another use case for --warmup is to collect profiling information for e.g. Class Data Sharing feature of JDK:

```
# Open JDK; launcher needs to be used because of simpler classpath
java -XX:DumpLoadedClassList=classes.lst -jar rootidr/launch-micro.jar  --warmup

# OpenJ9
java -Xshareclasses:name=payara-micro -jar payara-micro.jar --warmup
```

CAUTION	If using a CRaC JDK, the warmup option is instead used to activate creating a checkpoint during the bootstrap of Payara Micro. See "Payara Micro CRaC Support" for details.

Read Configuration from a File

With `--domainConfig` option, it is possible to define multiple options in a configuration file. This option would override the default Payara Micro configuration completely. The provided file must conform to the same structure as the `domain.xml` file in a Payara Server domain.

The `--rootDir` option sets the root configuration directory and saves the configuration across restarts. If empty, this directory will be filled by the default configuration, including the `domain.xml` file.

Precedence

If specifying multiple options at once, the following precedence is followed:

```
rootDir < domainConfig < autoBindHttp | autoBindSsl < port | sslPort
```

In human language:

- The *domain.xml* in the directory specified by the *rootDir* option (if one exists) is overridden by the *domain.xml* specified with the `domainConfig` option
- The HTTP and HTTPS port numbers specified in either of these *domain.xml* files are overridden to be the default values of **8080** and **8081** when the `autoBindHttp` or `autoBindSsl` options are enabled respectively.
- These default port values are then overridden in turn by the port numbers specified with the `port` or `sslPort` options.

Configuring an Instance Programmatically

There are various methods available for configuring a Payara Micro instance programmatically. You can only configure an instance before it is bootstrapped however.

The configuration methods available to you should be detected by your IDE, allowing you to view them using the auto-complete feature common to most popular IDEs. Alternatively, you can consult the "Payara Micro Configuration Methods" section.

As noted in the "Deploying an Application Programmatically" section, you can either call the desired configuration commands on one line during instance initialization, or on separate lines after creating a `PayaraMicro` variable.

As an example of configuring an instance to use a different HTTP and Cluster start port on one line, see here:

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicro;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
        PayaraMicro.getInstance().setHttpPort(2468).setClusterStartPort(5902
).bootStrap();
    }
}
```

For the example of the same, but done across multiple lines, see here:

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicro;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
        PayaraMicro micro = PayaraMicro.getInstance();
        micro.setHttpPort(2468);
        micro.setClusterStartPort(5902);
        micro.bootStrap();
    }
}
```

It is also possible to configure an instance programmatically by specifying a `domain.xml` file that is packaged within your application by passing a resource string to the `setApplicationDomainXML` method. The path in the string will be resolved using the `getResource` method of the thread context class loader:

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicro;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
        PayaraMicro.getInstance().setApplicationDomainXML("config/domain.xml"
).bootStrap();
    }
}
```

Packaging a Configured Instance as an Uber Jar

Sometimes it is preferable to package the application (or applications), configuration and dependencies into a single executable jar. To do this with Payara Micro use the `--outputUberJar` command line option as in this example:

```
java -jar payara-micro.jar --deploy test.war --outputUberJar test.jar
```

This will package up the `payara-micro.jar` and the WAR application into a single JAR. The resulting file can be executed like this:

```
java -jar test.jar
```

Any additional command line options you specify when creating an Uber JAR are recorded, so the Payara Micro instance is configured later when executing the packaged JAR:

```
java -jar payara-micro.jar --deploy test.war --port 9080 --lite --clusterName test-
cluster --clusterPassword test-password --outputUberJar test2.jar
```

IMPORTANT	All specified command line option will be retained when the Uber JAR is executed.

Uber JAR Context Root

When creating an Uber JAR, the context root of the packaged application will **always** be the name of the application WAR that is deployed. For example, the `test.war` that was packaged into the `test2.jar` on port **9080** would be accessible on the following path:

```
http://localhost:9080/test
```

Currently, this is **always** the case; including when a context root is specified in a `payara-web.xml`/`glassfish-web.xml` deployment descriptor.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

If the WAR file is renamed to `ROOT.war` and packaged as an Uber JAR, it will be deployed to the root context:

```
java -jar payara-micro.jar --deploy ROOT.war --port 9080 --outputUberJar test3.jar
java -jar test3.jar
```

The application will now be accessible on:

```
http://localhost:9080/
```

Package Additional Files

It's also possible to package additional files into an Uber JAR, by using a custom root directory. You can run a Payara Micro instance first by generating the domain directory first using the `--rootDir` option first:

```
java -jar payara-micro.jar --rootDir /tmp/micro-dir/
```

You can then add files to the root directory like this:

```
> cd /tmp/micro-dir/
> ls -lsarth

total 784K
   0 drwxr-xr-x 1 root 197609    0 Mar 24 18:16 docroot
160K -rw-r--r-- 1 root 197609 158K Mar 24 18:16 __ds_jdbc_ra.rar
   0 drwxr-xr-x 1 root 197609    0 Mar 24 18:16 META-INF
160K -rw-r--r-- 1 root 197609 159K Mar 24 18:16 __cp_jdbc_ra.rar
160K -rw-r--r-- 1 root 197609 159K Mar 24 18:16 __xa_jdbc_ra.rar
160K -rw-r--r-- 1 root 197609 160K Mar 24 18:16 __dm_jdbc_ra.rar
   0 drwxr-xr-x 1 root 197609    0 Mar 24 18:17 autodeploy
   0 drwxr-xr-x 1 root 197609    0 Mar 24 18:20 lib
4.0K drwxr-xr-x 1 root 197609    0 Mar 30 19:22 config
128K drwxr-xr-x 1 root 197609    0 Mar 30 19:23 runtime

> cp ~/test-properties /tmp/micro-dir/config/.
```

And then, generate the Uber JAR using the modified root directory:

```
java -jar payara-micro.jar --rootDir /tmp/micro-dir/ --outputUberJar custom-
micro.jar
```

You can verify that the files are located in the MICRO-INF/ directory:

```
> unzip -d custom-micro custom-micro.jar
> ls -lsarth custom-micro/** | grep

MICRO-INF/domain:
total 304K
1.0K -rw-r--r-- 1 fabio 197609   24 Apr  3 20:26 test.properties
```

IMPORTANT

Additional JAR files that are placed in the `${PAYARA_INSTALL_DIR}/lib` directory will be ignored when being packaged to the Uber JAR. To package additional JAR files into an Uber JAR, check the "Adding Third Party JARs section"

Configuring Payara Micro via System Properties and Environment Variables

Payara Micro can also be configured via system properties. These can either be set on the command line or passed into Payara Micro using the `--systemProperties` command line option which will load the properties from the specified file.

NOTE

Payara Micro can also be configured using Environment variables. The environment variables supported are the same as the system properties below just replace the `.` with `_` for example `payaramicro.port` should be `payaramicro_port` when used as a system property.

Payara Micro supports the following system properties:

System Property	Equivalent Command Line Flag
payaramicro.domainConfig	`--domainConfig`
payaramicro.hzConfigFile	`--hzConfigFile`
payaramicro.publicaddress	`--hzPublicAddress`
payaramicro.autoBindHttp	`--autoBindHttp`
payaramicro.autoBindRange	`--autoBindrange`
payaramicro.autoBindSsl	`--autoBindSsl`
payaramicro.logo	`--logo`
payaramicro.logToFile	`--logToFile`

System Property	Equivalent Command Line Flag
payaramicro.enableAccessLog	`--accessLog`
payaramicro.logPropertiesFile	`--logProperties`
payaramicro.enableDynamicLogging	`--enableDynamicLogging`
payaramicro.enableHealthCheck	`--enableHealthCheck`
payaramicro.port	`--port`
payaramicro.sslPort	`--sslport`
payaramicro.sslCert	`--sslcert`
payaramicro.mcAddress	`--mcAddress`
payaramicro.mcPort	`--mcPort`
payaramicro.hostAware	`--hostaware`
payaramicro.startPort	`--startport`
payaramicro.clusterName	`--clusterName`
payaramicro.clusterPassword	`--clusterPassword`
payaramicro.lite	`--lite`
payaramicro.maxHttpThreads	`--maxHttpThreads`
payaramicro.minHttpThreads	`--minHttpThreads`
payaramicro.noCluster	`--noCluster`
payaramicro.noHazelcast	`--noHazelcast`
payaramicro.disablePhoneHome	`--disablePhoneHome`
payaramicro.enableRequestTracing	`--enableRequestTracing`
payaramicro.requestTracingThresholdUnit	`--requesttracingthresholdunit`
payaramicro.requestTracingThresholdValue	`--requesttracingthresholdvalue`
payaramicro.rootDir	`--rootDir`
payaramicro.name	`--name`
payaramicro.instanceGroup	`--instancegroup`

System Property	Equivalent Command Line Flag
payaramicro.initialJoinWait	`--hzinitialjoinwait`

Configuring Alternate Keystores for SSL

Payara Micro comes with Keystore files directly embedded within the JAR file.

These can be overridden using the following standard Java SSL system properties:

- `javax.net.ssl.trustStore`
- `javax.net.ssl.keyStore`
- `javax.net.ssl.trustStorePassword`
- `javax.net.ssl.keyStorePassword`

IMPORTANT	When packaging applications into an Uber Jar any keystores specified via system properties will be copied into the uberjar to replace the default internal keystores. However the uber jar will not contain the passwords and these must still be specified via the system properties.
IMPORTANT	*asadmin* commands like `add-pkcs8` and `add-to-keystore` are NOT supported in pre-boot and post-boot commands with Payara Micro and Java SSL system properties need to be used to point to the correct stores.

Setting the Key-Pair at Runtime

The key-pair to use as Payara Micro's SSL certificate can also be specified using the `--sslCert` command line option like this:

```
java -jar payara-micro.jar -sslCert my-custom-alias
```

NOTE	The default key-pair name is `s1as` if not overridden via the command line switch.

Payara Micro Instance Names

Payara Micro instances are automatically assigned a name on boot. These names are generated from a dictionary of adjectives and fish names as `adjective-fish`. For example: `Magnanimous-Payara`, `Disgruntled-Goldfish`, and `Bamboozled-Tetra`.

Setting a Custom Instance Name

Instance names can be overridden with the `--name` command-line argument on start-up. Manually entered instances names are non-unique.

As an example, see below:

```
java -jar payara-micro.jar --name MicroInstance1
```

Payara Micro IPv4 and IPv6

Payara Micro instances automatically configure `java.net.preferIPv4Stack` to true shortly after launching unless configured otherwise on the command line. This is to match the default Hazelcast configuration of preferring IPv4, configured via `hazelcast.prefer.ipv4.stack`.

There may be situations where Payara Micro doesn't configure the `java.net.preferIPv4Stack` early enough, in which case it may be beneficial to define the JVM option on the command line yourself:

```
java -Djava.net.preferIPv4Stack=true -jar payara-micro.jar
```

If you wish Payara Micro and Hazelcast to prefer IPv6 addresses and an IPv6 stack, you can configure this using:

```
java -Djava.net.preferIPv4Stack=false -Djava.net.preferIPv6Addresses=true
-Dhazelcast.prefer.ipv4.stack=false -jar payara-micro.jar
```

Alternatively to return everything to the Java default (which isn't the Hazelcast default) of preferring an IPv6 stack and IPv4 address, you can configure Payara Micro and

Hazelcast using:

```
java -Djava.net.preferIPv4Stack=false -Dhazelcast.prefer.ipv4.stack=false -jar
payara-micro.jar
```


Deploying Applications

Deploying an Application from the Command Line

NOTE	If you wish to deploy multiple packages, you must specify the `--deploy` again, or point to a `--deploymentDir` with multiple applications.

Deploying an Application Package

NOTE	All Payara Micro actions are run for the Payara Micro JAR, all in one command.

To deploy a WAR file to an instance, you need to use the `--deploy` option, followed by the path to the application to deploy. See below for an example of starting a Payara Micro instance and deploying a WAR file:

```
java -jar payara-micro.jar --deploy /home/user/example.war
```

TIP	The `--deploy` option is optional if your deployment file ends with `.jar`, `.rar` or `.war` therefore the command line below can also be used

```
java -jar payara-micro.jar /home/user/example.war
```

Deploying an Exploded War

An exploded WAR can be deployed to a Payara Micro instance just by specifying the path to the exploded WAR root directory on the `--deploy` command line or programmatically via the API. The exploded WAR can be redeployed by creating an empty `.reload` file in the root directory and updating its timestamp.

TIP	On a Linux OS host, a simple `touch .reload` command can update the

> timestamp of the file.

Deploying Applications from a Maven Repository

You can deploy an application directly from a Maven repository using the `--deployFromGAV` option. This option accepts a comma separated string denoting a Maven artefact's `groupId`, `artifactId`, and `version` attributes.

```
java -jar payara-micro.jar --deployFromGAV "fish.payara.examples,test,1.0-SNAPSHOT"
```

This option can be used multiple times, and in conjunction with the standard `--deploy` options.

IMPORTANT	By default, Payara Micro will only search for artefacts in the Maven Central repository. If you wish to search additional repositories, you can add them to the list of repositories to search with the `--additionalRepository` option:

```
java -jar payara-micro.jar --deployFromGAV "fish.payara.examples,test,1.0-SNAPSHOT"
--additionalRepository https://maven.java.net/content/repositories/promoted/
```

To search through multiple additional repositories, you can use the option multiple times:

```
java -jar payara-micro.jar --deployFromGAV "fish.payara.examples,test,1.0-SNAPSHOT"
--additionalRepository https://maven.java.net/content/repositories/promoted/
--additionalRepository https://nexus.payara.fish/repository/payara-artifacts
```

Define Context Root

The context root of an application can be specified using the option `--contextroot` or by appending the path to the war-file name given with the `--deploy` option.

A context root set using `--contextroot <context-path>` option applies to the application of the first `--deploy` use that does not already specify a context root. Let's take the following example:

```
java -jar payara-micro.jar --deploy /home/user/example.war --contextroot hello-
world
```

The root context of the `example.war` application is now set to `/hello-world`.

Alternatively the context root can be included in the value of the `--deploy` option. It follows the path to the war file separated by `File.pathSeparator`:

- `--deploy <war-file>:<context-path>` (Linux uses `:`)
- `--deploy <war-file>;<context-path>` (Windows uses `;`)

```
java -jar payara-micro.jar --deploy /home/user/example.war:hello-world
```

The above command again results in the effective root context `/hello-world`.

The two alternative mechanisms can also be used together:

```
java -jar payara-micro.jar --contextroot bar
    --deploy /home/user/example.war:foo
    --deploy /home/user/test.war
```

Here `example.war` is deployed in `/foo` while `test.war` is deployed in `/bar`.

NOTE `--deploy example.war:` is identical to `--deploy example.war:/` and deploys to the effective context root `/` while `--deploy example.war` deploys to `/example`.

IMPORTANT It follows that leading `/` can be given but is not required. However, tailing `/` must not be given. Using `--deploy example.war:/foo/` will render the `<host>/foo/` endpoint inaccessible.

Define Global Context Root

The global context root for all applications can be specified using the option `--globalcontextroot`.

Let's take the following example:

```
java -jar payara-micro.jar --deploy /home/user/example.war --contextroot hello-
world --globalcontextroot api
```

The complete context of the `example.war` application is now set to `/api/hello-world`.

IMPORTANT	It follows that leading `/` can be given but is not required. However, tailing `/` must not be given.

Using Root Configuration Directory

When a configuration root directory is reused across restarts (i.e. not running in a container), it is recommended not to set up a deployment each time a Payara Micro instance is started using that directory.

Doing so will cause the application to be deployed twice: Once at startup and second time when deployment section of the command line is processed.

The best approach is to use a warmup run for deployment the first time the root directory is created:

```
# Perform deployments and configuration the first time
java -jar payara-micro.jar --rootdir micro-root --warmup application.war

# All further invocations do not setup deployment
java -jar payara-micro.jar --rootdir micro-root
```

Alternatively a launcher can be generated from a configuration root directory as well:

```
# Perform deployments and configuration the first time
java -jar payara-micro.jar --rootdir micro-root --warmup application.war
java -jar payara-micro.jar --rootdir micro-root --outputlauncher

# All further invocations use the launcher JAR in the configuration directory
java -jar micro-root/launch-micro.jar
```

Payara Micro Post Boot Output

The following information about the output made by Payara Micro after its booting has

completed will be available for examination:

Deployed Apps

After Payara Micro has booted, the log output will list data about the instance and all deployed applications, which includes:

- The name of the host machine
- The HTTP port(s) in use by the instance
- The HTTPS port(s) in use by the instance
- The generated instance name
- The instance's group name
- The Hazelcast member UUID
- A list of all deployed application names and their entry-point URLs

Here is an example of this output:

```
[DATE:TIME+TIMEZONE] [] [INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main]
[timeMillis: EPOCHTIME] [levelValue: 800] [[[DATE:TIME+TIMEZONE] [] [INFO] []
[PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: EPOCHTIME]
[levelValue: 800] [[
Instance Configuration
Host: Example-Host
HTTP Port(s): 8080
HTTPS Port(s):
Instance Name: Adjective-Fish
Instance Group: MicroShoal
Hazelcast Member UUID 000a0aa0-000a-0a00-000a-a00000a00aa0
Deployed: ExampleApp01 ( exampleapp01 war /exampleappendpoint1 )
Deployed: ExampleApp02 ( exampleapp02 war /exampleappendpoint2 )
]]

[DATE:TIME+TIMEZONE] [] [INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main]
[timeMillis: EPOCHTIME] [levelValue: 800] [[[DATE:TIME+TIMEZONE] [] [INFO] []
[PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: EPOCHTIME]
[levelValue: 800] [[

Payara Micro URLs
http://example:port/exampleappendpoint1
https://example:port/exampleappendpoint1
http://example:port/exampleappendpoint2
https://example:port/exampleappendpoint2
]]
```

Deploying Applications on Micro Programmatically

You can deploy applications onto Payara Micro programmatically via its API.

Deploying Applications Using an Asadmin Command

Applications can be deployed using the equivalent administration (`asadmin`) commands present on Payara Server.

Deploying on a Local Instance

It's possible to use the `run(Collection<InstanceDescriptor> members, String command, String… args )` method to run the `deploy` asadmin command. To deploy to a single instance, you must create a `members Collection` object only containing a single instance like in the following example:

```
import fish.payara.micro.PayaraMicro;
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicroRuntime;
import fish.payara.micro.services.data.InstanceDescriptor;
import java.util.ArrayList;
import java.util.Collection;

public class EmbeddedPayara {
    public static void main(String[] args) throws BootstrapException {
        PayaraMicroRuntime runtime = PayaraMicro.getInstance().setHttpAutoBind(
true).bootstrap();

        // Get a Collection with all running instances in the local runtimes
cluster
        Collection<InstanceDescriptor> allInstances = runtime.
getClusteredPayaras();

        // Create a Collection to use as a subset
        Collection<InstanceDescriptor> subset = new ArrayList<>();

        // Loop through all instances
        for (InstanceDescriptor instance : allInstances){
            // Only add the local instance
            if (instance == runtime.getLocalDescriptor()){
                subset.add(instance);
                break;
            }
        }

        // Run the deploy asadmin command on the subset Collection
        runtime.run(subset, "deploy", "/home/user/test.war");
    }
}
```

Deploying on Multiple Running Instances

You can use the `run` method to run the `deploy` asadmin command on multiple Payara Micro instances that belong to the same cluster. There are two `run` method variants available for running administration commands: One which runs an `asadmin` command on a subset of instances in a cluster, and another than runs an `asadmin` command on ALL instances in a cluster.

Here is an example of deploying an application to ALL instances in a cluster:

```
import fish.payara.micro.PayaraMicro;
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicroRuntime;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
        PayaraMicroRuntime runtime = PayaraMicro.getInstance().setHttpAutoBind(
true).bootstrap();

        // Run the deploy asadmin command on all instances in the runtimes cluster
        runtime.run("deploy", "/home/user/test.war");
    }
}
```

See the example for deploying on local instance on how to use the `run` method on a subset of instances in a cluster.

Deploying an Application Programmatically During Bootstrap

There are two methods you can use to deploy an application during Payara Micro's bootstrap process:

- `addDeployment(String pathToWar)`
- `addDeploymentFile(File file)`

The first, `addDeployment(String pathToWar)`, accepts a String that points to the path of the artefact to be deployed. For example:

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicro;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
        PayaraMicro.getInstance().addDeployment("/home/user/example.war").
```

```
bootStrap();
    }
}
```

The second method, `addDeploymentFile(File file)`, functions in the same way, but takes a `File` object instead:

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicro;
import java.io.File;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
        File file = new File("/home/user/example.war");
        PayaraMicro.getInstance().addDeploymentFile(file).bootStrap();
    }
}
```

Unlike when controlling Payara Micro from the command line, you can split the instance initialization and configuration across multiple method invocations. For example, to deploy an application and start the instance on separate method calls, you can run the following code:

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicro;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
        PayaraMicro micro = PayaraMicro.getInstance();
        micro.addDeployment("/home/user/example.war");
        micro.bootStrap();
    }
}
```

TIP You can use this strategy to deploy multiple deployments by calling `addDeployment` multiple times, or use `setDeploymentDir` to point to a deployment directory that contains the applications to deploy.

Deploying an Application Programmatically after Bootstrap

There are three methods available for deploying an application to a running instance:

- `deploy(File war)`

- `deploy(String name, InputStream is)`
- `deploy(String name, String contextRoot, InputStream is)`

`Deploy(file War)`

The first method works in the exact same way as the `addDeploymentFile` method described in the Deploying Programmatically during Bootstrap section.

`Deploy(string Name, Inputstream Is)`

This method allows you to deploy WAR files using an `InputStream`:

```
import fish.payara.micro.PayaraMicro;
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicroRuntime;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.logging.Level;
import java.util.logging.Logger;

public class EmbeddedPayara {
    public static void main(String[] args) throws BootstrapException{
        PayaraMicroRuntime instance = PayaraMicro.bootstrap();

        try (InputStream is = new FileInputStream("/home/user/test.war")){
            instance.deploy("test", is);
        }
        catch (IOException ex){
            Logger.getLogger(EmbeddedPayara.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
}
```

`Deploy(string Name, String Contextroot, Inputstream Is)`

This method works in the same way as the previous one, but allows you to specify the context root of the application at the same time. The following example would deploy the `test` application to a context root of `/app`:

```
import fish.payara.micro.PayaraMicro;
import fish.payara.micro.BootstrapException;
import fish.payara.micro.PayaraMicroRuntime;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.logging.Level;
import java.util.logging.Logger;

public class EmbeddedPayara{
```

```
    public static void main(String[] args) throws BootstrapException{
        PayaraMicroRuntime instance = PayaraMicro.bootstrap();

        try (InputStream is = new FileInputStream("/home/user/test.war")){
            instance.deploy("test", "app", is);
        }
        catch (IOException ex){
            Logger.getLogger(EmbeddedPayara.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
}
```

Deploying an Application Programmatically from a Maven Repository

Payara Micro supports the deployment of an application's artefact that is hosted in a Maven repository.

Deploying an Artefact Using Maven Coordinates

To deploy an application directly from a Maven repository, you will need to add a Maven GAV coordinate. This can be done using `addDeployFromGAV()` method. This method accepts a comma separated string denoting a maven artefact's `groupId`, `artifactId`, and `version` attributes.

```
import fish.payara.micro.PayaraMicro;
import fish.payara.micro.BootstrapException;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
       PayaraMicro.getInstance().addDeployFromGAV("fish.payara.testing,clusterjsp,
1.1").bootStrap();
    }
}
```

Specifying Additional Maven Repositories

By default, Payara Micro will only search for artefacts in the Maven Central repository. If you wish to search additional repositories, you can add them to the list of repositories to search with the `addRepoUrl()` method:

```
import fish.payara.micro.PayaraMicro;
import fish.payara.micro.BootstrapException;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
       PayaraMicro.getInstance().addRepoUrl
("https://nexus.payara.fish/repository/payara-artifacts");
```

```
        PayaraMicro.getInstance().addDeployFromGAV
("fish.payara.testing,clusterjsp,1.1").bootStrap();
    }
}
```

To search through multiple additional repositories, you can simply call the `addRepoUrl` method, and specify a comma-separated list of the URLs of each repository:

```
import fish.payara.micro.PayaraMicro;
import fish.payara.micro.BootstrapException;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
        PayaraMicro micro = PayaraMicro.getInstance();
        micro.addRepoUrl("https://nexus.payara.fish/repository/payara-artifacts",
"https://maven.java.net/content/repositories/promoted/");
        micro.addDeployFromGAV("fish.payara.testing,clusterjsp,1.1");
        micro.bootStrap();
    }
}
```

Deploying Multiple Applications from a Maven Repository

To deploy multiple applications hosted in a Maven repository, you must call the `addDeployFromGAV` method for each application.For example, to deploy two applications:

```
import fish.payara.micro.PayaraMicro;
import fish.payara.micro.BootstrapException;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
        PayaraMicro micro = PayaraMicro.getInstance();
        micro.addDeployFromGAV("fish.payara.testing,clusterjsp,1.1");
        micro.addDeployFromGAV("fish.payara.testing,demo,1.0");
        micro.bootStrap();
    }
}
```

Jar Structure and Configuration

Payara Micro JAR Structure

This page details the structure of the currents Payara Micro JAR file.

Payara Micro JAR Folder Structure

The following is an overview of the file structure for a Payara Micro JAR file:

```
payara-micro.jar
¬¬¬ fish
¬   ¬¬¬ payara/micro/
¬¬¬ META-INF
¬   ¬¬¬ MANIFEST.MF
¬   ¬¬¬ maven/fish.payara.micro/payara-micro-boot/pom.xml
¬¬¬ MICRO-INF
    ¬¬¬ classes
    ¬¬¬ deploy
    ¬¬¬ domain
    ¬¬¬ lib
    ¬¬¬ payara-boot.properties
    ¬¬¬ post-boot-commands.txt
    ¬¬¬ pre-boot-commands.txt
    ¬¬¬ runtime
```

Structure Descriptions

Here's a brief summary of the folders and files that belong to the structure shown earlier:

File	Description
`fish/payara`	Payara Micro's class files.
`META-INF`	Contains the **Manifest** and **POM** files.
`MICRO-INF/classes`	Contains classes which are added to the class path before those in the `/runtime` folder.
`MICRO-INF/deploy`	Contains WAR, EAR, and EJB-JAR files for deployment.
`MICRO-INF/domain`	Contains the *domain.xml*, *default-web.xml*, keystores, *login.conf*, *logging.properties* files, and other files that are written to the temporary file directory.
`MICRO-INF/lib`	Contains additional third party dependency jars which will be automatically added to the instance's classpath.
`MICRO-INF/runtime`	Contains the core runtime jars.
`MICRO-INF/payara-boot.properties`	The System properties file containing Payara Micro runtime flags. This will override runtime flags and can be overridden by command-line arguments.

File	Description
`MICRO-INF/<command-files>.txt`	**.txt** files containing asadmin commands to execute post boot can also be stored within MICRO-INF. This includes `pre-boot-commands`, `post-boot-commands`, and `post-deploy-commands`.

Nested JAR

Payara Micro has two options for unpacking classes:

Unpacking to File System (`--Unpack`)

By default, Payara Micro will unpack the nested JARs into a temporary directory within the directory specified by either the system property `java.io.tmpdir` or the command line argument `--unpackdir`, and then load them as classes.

Unpacking to Memory (`--Nested`)

The `--nested` argument will load the classes directly from the nested JARs to the memory without unpacking the JARs into a folder, but may slow the booting process.

To start Payara Micro as a nested JAR, use the `--nested` option as shown:

```
java -jar payara-micro.jar --nested
```

Adding Third-Party Jars to a Micro Instance

There are multiple scenarios when it's needed to add a third-party library to the class path of a Payara Micro instance:

- Deploying applications that use JDBC resources, which have a need to access the respective JDBC driver JAR.
- Shared libraries used by multiple applications deployed on the same instance.
- Avoiding the creation of fat application WARs with lots of bundled libraries to simplify a build-integration pipeline.

Such scenarios are easily solved on Payara Server by adding these JAR files at the domain or modules of the application server library directories. On Payara Micro, this

can be done on multiple ways.

From the Command Line

In order to add a third-party jar to a new Payara Micro instance, use the `--addLibs` argument when starting the instance like this:

```
java -jar payara-micro.jar --addLibs third-party.jar
```

You can specify a directory instead of a simple JAR to add all the JARs that are located in that directory:

```
java -jar payara-micro.jar --addLibs /home/payara-libs/
```

Finally, in order to add third-party libraries from multiple sources, specify each source (either a file or directory path) separating them using the colon character (`:`). Here's a simple example to add 3 sources for libraries:

```
java -jar payara-micro.jar --addLibs third-
party.jar:/opt/application/jars:~/v/mysql-jdbc-driver.jar
```

NOTE You may notice that this syntax is similar to the syntax used to specify the classpath using the `-cp` argument when running the JVM.

If you don't like using the previous syntax and prefer a simpler option, you can repeat the `--addLibs` argument as many times as you want per each JAR source, like in the following example:

```
java -jar payara-micro.jar --addLibs third-party.jar --addLibs
/opt/application/jars --addLibs ~/v/mysql-jdbc-driver.jar
```

This argument cannot be used when instance is started via "root directory launcher".

It is however accepted when creating a launcher via `--outputlauncher` and provided libraries will be added to launcher's classpath.

Programmatically Using the API

In order to add third-party libraries to a new Payara Micro instance programmatically, use the `PayaraMicro.addLibrary` method to properly add either JAR files or a directory of JAR files. You can call the method multiple times to each multiple sources.

For example, to add both a JAR file and a new directory of JAR files to a new instance use the following code:

```
public static void main(String[] args) throws BootstrapException {
    PayaraMicro instance = PayaraMicro.getInstance();
    instance.addLibrary(new File("~/my-library"));
    instance.addLibrary(new File("~/my-libs/"));
    instance.bootStrap();
}
```

WARNING

Unlike other configuration methods of the `PayaraMicro` class, the `addLibrary` method **DOES NOT RETURN** a reference to the `PayaraMicro` object, which means that this method cannot be used for method-chaining style configuration. This is an issue which will be solved on a future release.

Packaging Libraries in an Uber JAR

When creating an Uber JAR using the `--outputUberJar` argument from the command line, all specified third-party libraries will be packaged on the resulting JAR. All added JARS will be located on the `MICRO-INF/lib` directory.

Here is an example to package 2 JDBC drivers (for MySQL and PostgreSQL) into an Uber JAR called `my-micro.jar`:

```
java -jar payara-micro.jar --addLibs postgresql.jar:mysql-connector-java.jar
--outputUberJar my-micro.jar
```

We can inspect this Uber JAR to verify that the third-party JAR files were packaged successfully:

```
> jar -tf my-micro.jar | grep "MICRO-INF/lib"
```

```
MICRO-INF/lib/
MICRO-INF/lib/postgresql.jar
MICRO-INF/lib/mysql-connector-java.jar
```

Root Configuration Directory

Payara Micro is distributed as a single JAR file. However, at runtime, it creates and depends on a folder structure that is referred to as Root Configuration Directory or `rootdir`.

This directory stores all of the instance configuration, deployed applications, as well as implementation JAR files needed for running the instance.

Temporary Root Configuration Directory

By default, Payara Micro will create a folder in the system's temporary directory and prepares a root configuration directory there. This makes instances' configuration stateless, but also has two potentially undesirable effects:

- Temporary directory may be deleted by system cleanup procedure. It is therefore important not to use a temporary directory in production scenarios. If files are removed by a third party process, the application will not behave correctly anymore.
- Preparing the directory has an impact on startup time

Persistent Root Configuration Directory

Root configuration directory can be explicitly specified with the command line parameter `--rootDir <directory>`. In such case, the directory is created only once, and instance configuration is preserved across restarts of the instance.

Following configuration is preserved:

- Deployed applications
- Effects of preboot, postboot and post-deploy commands
- Logging configuration

Clustering

A Payara Micro cluster is based on Hazelcast, similarly to how Hazelcast clustering

works in Payara Server. Each node in a Payara Micro cluster is effectively a Hazelcast node.

This makes it easy to dynamically scale infrastructure setup, without any complex configuration or downtime required.

Lite Cluster Members

You may configure a Payara Micro instance as a Hazelcast *lite* node, meaning that the instance will join an existing cluster but will not store any data; like web session data or JCache data, for example.

To do this, use the `--lite` option when launching the instance through the command line.

You can create a cluster topology whereby a web application is hosted in a number of Payara Micro instances and the garbage collection ergonomics for these instances are tuned for throughput. In addition, a number of Payara Micro instances can be in the cluster with no applications deployed and these instances can be tuned for long-lived web session data. In this case, the web application instances could be designated as lite cluster members to make sure no web session data is stored within their JVMs.

IMPORTANT	Lite members can also be used purely if you want a clustered Payara Micro instance to join the same cluster and receive CDI events or clustered events, but without storing any data.

Automatic Clustering

Payara Micro has the ability to auto-cluster, since by default it uses the Multicast discovery mode.

Discovery Modes

Like in Payara Server, the following discovery modes are supported in Payara Micro:

Multicast Discovery Mode

The integration of Hazelcast into Payara Micro allows instances to automatically and

dynamically cluster together when two instances or more instances are pointed at the same multicast address and port.

Example

Start two Payara Micro instances (with different HTTP port configurations or use `--autobindhttp`), and you should see the following output:

```
Members [2] {
Member [192.168.174.130]:5900 this
Member [192.168.174.130]:5901
```

IMPORTANT	The `--startPort` option is used to determine which clustering port the Payara Micro instance will **first** try to bind to. If the port is already bound to then the Payara Micro instance will simply increment the *startPort* value and try again until it finds a port it can bind to.

Preventing Multicast Cluster Cross Talk

By default Payara Micro clusters automatically discover other cluster members via multicast. This can lead to the situation whereby different development environments that are used by different teams may unintentionally cluster together as they are using the same multicast address and multicast port.

If you wish to separate multiple clusters, then you can use the `--mcAddress` and `` `--mcPort `` options to define a different multicast address and port; assign a different address and port to each set of instances you wish to run in a separate cluster and they will automatically make their own cluster on the new multicast address and port.

TIP	You can also use `--clusterName` and `--clusterPassword` to separate clusters. If all other multicast settings are similar, instances will only cluster together if all the instances have the same cluster name and cluster password.

Domain Discovery Mode

Domain Discovery Mode enables a Payara Micro instance to join an existing Domain Data Grid managed by a Payara Server DAS. To join a Domain Data Grid start the Micro instance in domain cluster mode:

```
java -jar payara-micro.jar --clustermode domain:192.168.0.64:4900
```

The format of the domain cluster mode is `domain:ip:port` where `ip` is the IP network address that the Hazelcast instance is bound to in the DAS.

NOTE Hazelcast is very particular about the IP address used to join a cluster. The IP Address should be exactly as reported by the details

When the DAS starts it will log output similar to this one:

```
Data Grid Status
Payara Data Grid State: DG Version: 35 DG Size: 2
Instances: {
Address: /192.168.0.104:4900 UUID: 7c1bd6ca-1574-4e38-9189-effcd4ccafd2 Lite: false
This: true Name: server Group: server-config
Address: /192.168.0.104:5900 UUID: 0f6887b2-9932-4085-a6f8-e40751269023 Lite: false
This: false
}
```

This shows the IP address and ports that can be used as input into the domain cluster mode.

NOTE If you accidentally choose the DAS admin port (by default `4848`) as the cluster port you will get a warning like the following:

```
[2018-03-08T13:51:46.446+0000] [] [WARNING] []
[fish.payara.nucleus.hazelcast.DomainDiscoveryService] [tid: _ThreadID=1
_ThreadName=main] [timeMillis: 1520517106446] [levelValue: 900] You have specified
4848 as the datagrid domain port however this is the default DAS admin port, the
default domain datagrid port is 4900
```

TCPIP Discovery Mode

The TCPIP Discovery Mode can be used to cluster Payara Micro instances when multicast is not available on the network. This is especially useful in Cloud

environments or in containerized deployments like Docker and Kubernetes. For the TCPIP discovery mode you specify a comma separated list of `ip:port` addresses where you believe other instances will be listening.

NOTE	Hazelcast is very particular about the IP address used to join a cluster. The IP Address should be exactly as reported by the details

For example the following command will cluster two Payara Micros together when multicast is not available:

```
java -jar payara-micro.jar --autobindhttp --clustermode
tcpip:192.168.0.104:6900,192.168.0.104:6901
java -jar payara-micro.jar --autobindhttp --clustermode
tcpip:192.168.0.104:6900,192.168.0.104:6901
```

A range of IP addresses can be specified in the list of members for example: `192.168.1.0-7:6900`. This is especially useful if you have a cloud VPC and you have a number of Payara Micro instances running across a number of Cloud VMs. Using a range is also useful in Docker environments where the range of IP Addresses is known but dynamic for example `172.17.0.2-8:6900`.

During discovery Payara Micro will test each `IP:Port` combination until it finds another Payara Micro instance. As long as one instance exists then it will join the existing cluster. If no Payara Micro instances are contactable the instance will create a new cluster with itself as the single member.

NOTE	TCPIP cluster mode can also be used to join an existing Payara Server Domain Data Grid by specifying the `ip:port` of the DAS Hazelcast's instance.

DNS Discovery Mode

The DNS Discovery mode can be used to cluster Payara Micro instances by specifying a DNS name. Similar to the TCPIP discovery mode, this discovery mode takes a comma separated list of DNS **A/AAAA** record names and ports to determine where to look for

instances to cluster with, like this:

```
java -jar payara-micro.jar --autobindhttp --clustermode
dns:localhost:6900,ubuntu:6900,suse:6900
```

IMPORTANT	If there are multiple IP addresses associated with a DNS name, Payara Micro will attempt to cluster with all of them.

Kubernetes Discovery Mode

The Kubernetes discovery mode can be used to cluster Payara Micro instances running in a Kubernetes cluster, which can be useful due to the variable networking inherent to the technology. When using this discovery mode, the default settings will make Payara Micro attempt to cluster with instances present on Pods within the default namespace:

```
java -jar payara-micro.jar --autobindhttp --clustermode kubernetes
```

Optionally, you can provide a Kubernetes namespace and service name separated by a comma. Payara Micro will then attempt to cluster with instances present on pods within the specified namespace and service. If using the latter option, you must provide BOTH a namespace and service name, as these options cannot be used individually.

```
java -jar payara-micro.jar --autobindhttp --clustermode kubernetes:default,service1
```

This discovery mode internally uses the Hazelcast Kubernetes plugin, and so you will need to grant specific permissions to query the Kubernetes master node. As per the plugin documentation, you will need to apply the following permissions using the `kubectl apply` command:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: default-cluster
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: view
subjects:
- kind: ServiceAccount
```

```
  name: default
  namespace: default
```

Network Interfaces

If Hazelcast chooses the incorrect IP Address to bind to (for example a docker interface) the `--interfaces` command line option can be used to force Hazelcast to use the specified network interface and prevent second-guesses:

```
java -jar payara-micro.jar --autobindhttp --clustermode
tcpip:192.168.0.104:6900,192.168.0.104:6901 --interfaces 192.168.0.104
```

See Also

- Grating Permissions To Use Kubernetes API
- Hazelcast Configuration

HTTP and HTTPS Auto-Binding

Payara Micro supports an auto-binding feature for its HTTP and HTTPS port, allowing Payara Micro to dynamically find a free HTTP or HTTPS port as required during startup.

This feature is controlled through these three options:

`--autoBindHttp`: Enables auto-bind functionality for the HTTP port **`--autoBindSsl`**: Enables auto-bind functionality for the HTTPS port **`--autoBindRange`**: Sets the range for the auto-bind feature

When auto-binding is enabled, Payara Micro will attempt to connect to the port specified with the `--port` or `--sslPort` option (or, if these options are not specified, to port **8080** and **8181** respectively), and if that port is busy, it will instead try to bind on the next port up, repeating this process until it finds a free port, or the `--autoBindRange` limiter is reached.

For example, if ports **8080** and **8081** were busy, and ports **8082** and **8182** were free, then starting Payara Micro with the following command would have the HTTP port bind to **8082** and the HTTPS port bind to **8182**:

```
java -jar payara-micro.jar --autoBindHttp --sslPort 8181 --autoBindSsl
```

Whereas the following would fail to start in the same situation because `--autoBindRange` is not large enough (no free port within the range of **8080-8081**):

```
java -jar payara-micro.jar --port 8080 --autoBindHttp --sslPort 8181 --autoBindSsl
--autoBindRange 1
```

WARNING

Be aware that the auto-bind feature does not currently read port values from domain.xml files; if the `--port` and `--sslPort` options are not used, then the `--autoBindHttp` and `--autoBindSsl` options will assume that the HTTP and HTTPS ports will be at the default values of 8080 and 8181 respectively.

Command Line Options

Payara Micro Command Line Options

All of the options listed below are case-insensitive starting from release `4.1.1.171`; prior versions need the option to be set with camelCase notation

Configuration Option	Description	Default Value
`--accesslog <directory-path>`	Sets user defined directory path for the access log.	
`--accesslogformat`	Sets user defined log format for the access log.	"%client.name% %auth-user-name% %datetime% %request% %status% %response.length%"
`--accessloginterval`	Sets user defined access log write interval.	300
`--accesslogsuffix`	Sets user defined simple date time suffix for access log filenames.	`yyyy-MM-dd`
`--accesslogprefix`	Sets user defined prefix for access log filenames.	

Configuration Option	Description	Default Value
`--additionalrepository <repo-url>`	Specifies an additional repository to search for deployable artefacts in. Supports BASIC authentication. For example https://USER:PASSWORD@example.payara.fish/repositories/repo/. You may also use aliases https://USER:{ALIAS=ALIAS_NAME}@example.payara.fish/repositories/repo/	
`--addjars <jar-path>`	*Deprecated*. Use `--addlibs` instead	
`--addlibs <file or directory path>`	<file-or-directory-path> add specified jar files into classloader, or packaged into an Uber JAR. Multiple files can be specified as colon separated values.	
`--autobindhttp`	Enables auto-binding for the HTTP port.	*false*
`--autobindrange <number-of-ports>`	Sets the range for HTTP and HTTPS port auto-binding.	5
`--autobindssl`	Enables auto-binding for the HTTPS port.	*false*
`--clustermode`	Modifies the cluster discovery mode of Payara Micro when finding members of the cluster. Options are: • `multicast:group:port` • `tcpip:host:port` • `domain:dashost:port`	multicast
`--clustername <cluster-name>`	Sets the cluster group name.	
`--clusterpassword <cluster-password>`	Sets the cluster group password.	

Configuration Option	Description	Default Value
`--contextroot <context-path>`	Sets the context root of the application originating from the `--deploy` option given first that does not specify a context root itself. See define-context-root for more details.	
`--copytouberjar <directory-path`	<directory-path> copy a directory into the root of an uber JAR	
`--deploy <file-path>[:<context-path]`	Specifies a WAR file to deploy and optionally the context path to use for that application.	If not set, has no value and is not used.
`--deploydir <directory-path>`	Sets a directory which will be scanned for EAR, WAR, and RAR files for auto-deployment.	If not set, has no value and is not used.
`--deployfromgav <list-of-artefacts>`	Specifies a comma separated `groupId, artifactId, versionNumber` of an artefact to deploy from a repository.	
`--deploymentdir <directory-path>`	See `--deploydir`.	
`--disablephonehome`	Disables the *Phone Home* service for this instance.	If not set, the *Phone Home* service is active.
`--domainconfig <file-path>`	Overrides the server configuration with an alternative *domain.xml* file.	If not set, the domain.xml contained in the Payara Micro JAR is used.
`--enableDynamicLoggi ng`	Watch the logging properties file for changes, and apply them dynamically	
`--enablehealthcheck <boolean>`	enables/disables the HealthCheck Service.	*false*
`--enablerequesttraci ng`	Enables the Request Tracing Service and optionally sets the threshold unit and/or value.	

Configuration Option	Description	Default Value
`--enablerequesttracingadaptivesampling <boolean>`	<boolean> Enables the adaptive sampling. Can also be enabled by configuring any one of the adaptive sampling configuration options	
`--globalcontextroot <context-path>`	Sets the global context root of any deployed application. This will prepend the context root. See define-global-context-root for more details.	
`--group <group-name>`	Sets the instance's group.	
`--help`	Displays the configuration options and then exits.	If not set, this option is not used.
`--hostaware`	Enable Host Aware Partitioning. When enabled cluster backups will not be stored on members with the same IP Address.	*false*
`--hzconfigfile <file-path>`	Sets the Hazelcast configuration file to use for overriding the built-in Hazelcast cluster configuration.	If not set, the built-in Hazelcast configuration file is used.
`--hzinitialjoinwait <time in seconds>`	Sets the time in seconds for Payara Micro to wait before attempting to join the data grid.	1
`--hzPublicAddress <host-name or ip>`	The Public Address to use for this Data Grid Node. Used when the node is behind a NAT	
`--instancegroup <group-name>`	See `--group`	
`--interfaces`	Specifies the Interfaces that the data grid discovery mode should use to listen. Can be comma separated, e.g. 127.0.0.1,192.168.0.*	

Configuration Option	Description	Default Value
`--lite`	Sets the micro container to lite mode, which means it clusters with other Payara Micro instances but does not store clustered data.	*false*
`--logo`	Reveals either the **#BadAssFish** or a custom logo on boot.	
`--logproperties <file-path>`	Allows user to set their own logging properties file.	
`--logtofile <file-path>`	Outputs all the Log entries to a user defined file.	
`--maxhttpthreads <max-thread-count>`	Sets the maximum number of threads in the HTTP thread pool.	10
`--mcaddress <multicast-address>`	Sets the cluster multicast group if multicast cluster mode is selected. See `--clustermode`.	
`--mcport <multicast-port>`	Sets the cluster multicast port if multicast cluster mode is selected. See `--clustermode`.	2904
`--minhttpthreads <min-thread-count>`	Sets the minimum number of threads in the HTTP thread pool.	10
`--name <instance-name>`	Sets the instance name.	Generated Universally Unique Identifier *(Adjective-Fish)*.
`--nested`	Do not unpack the Nested Jars when booting the server. This is generally slower than unpacking the runtime.	*false*
`--nocluster`	Disables clustering for this instance.	*false*
`--noHazelcast`	Disables Hazelcast and clustering for this instance.	*false*

Configuration Option	Description	Default Value
`--nohostaware`	Disables Host Aware Partitioning. See `--hostaware`	
`--outputuberjar <file-path>`	Packages up an Uber JAR at the specified path based on the provided command line arguments and exit.	
`--outputlauncher`	Create launcher `launch-micro.jar` into the root directory that is specified by `--rootdir` and exit.	
`--port <http-port-number>`	Sets the HTTP port that the instance will bind to.	8080
`--postbootcommandfile <file-path>`	Provides a file of asadmin commands to run **after booting the server**.	
`--postdeploycommandfile <file-path>`	Provides a file of asadmin commands to run **after all deployments have completed**.	
`--prebootcommandfile <file-path>`	Provides a file of asadmin commands to run **before booting the server**.	
`--requesttracingadaptivesamplingtargetcount`	The target number of traces to sample per the configured time window	6
`--requesttracingadaptivesamplingtimeunit`	The time unit for the adaptive sample time; a `java.util.concurrent.TimeUnit` value (also in singular) or one of the short forms: `ns`, `us`/`µs`, `ms`, `s`, `m`/`min`/`mins`, `h` or `d`.	MINUTES
`--requesttracingadaptivesamplingtimevalue`	The period of time to attempt to hit the adaptive sample target count in	1
`--requesttracingthresholdunit <threshold-unit-notation>`	Sets the time unit for the requestTracingThresholdValue option, i.e. `SECONDS`, `ms`, `days` etc.	

Configuration Option	Description	Default Value
`--requesttracingthresholdvalue <threshold-unit-value>`	Sets the threshold time before a request is traced.	
`--rootdir <directory-path>`	Sets the root configuration directory and saves the configuration across restarts.	Defaults to `java.io.tmpdir`.
`--secretsdir`	Directory to read configuration from key-value files using the Microprofile config api.	
`--shutdowngrace <duration-ms>`	After server shutdown is initiated the server waits this period of time to finish ongoing requests before terminating. Currently, new requests are still accepted during this grace period.	0
`--sslport <ssl-port-number>`	Sets the HTTPs port that the instance will bind to.	If not set, has no value and HTTPS is disabled.
`--sslcert <certificate-name>`	Sets the SSL certificate name to be used from the keystore for SSL	`s1as`
`--startport <cluster-start-port-number>`	Sets the cluster start port number.	5900
`--systemproperties <file-path>`	Reads system properties from a file.	
`--unpackdir <directory-path>`	Unpack the Nested Jar runtime jars to the specified directory.	Unpack to `java.io.tmpdir`.
`--version`	Displays the version information	
`--warmup`	Exit the server immediately after configuration is done, applications are deployed and it's ready to serve requests. If using a CRaC JDK, this option is instead used to activate creating a checkpoint during the bootstrap of Payara Micro. See "Payara Micro CRaC Support" for details.	

Disable Phone Home in Payara Micro

Payara Micro, like Payara Server, has the Phone Home functionality enabled by default. It can be easily disabled in Payara Micro.

Disable Phone Home for Payara Micro

When starting Payara Micro, the Phone Home Service can be disabled with the command-line argument `--disablephonehome`:

```
java -jar payara-micro.jar --disablephonehome
```

See Also

- phone-home

Asadmin Commands

Running Asadmin Commands Using Pre-Boot and Post-Boot Scripts

Administration commands can also be run against a Payara Micro instance at specific moments in its lifecycle. To do this, you must provide the commands you wish to execute in special scripts via a command line option.

Structure of a Boot Script

Boot scripts are simple text files that contain the *asadmin* subcommands used to configure the instance. These scripts will follow these set of rules:

1. Each *asadmin* command must be represented in one line of text in the script. No line breaks are allowed.
2. Each *asadmin* command must be formatted on the syntax `subcommand [options] [operands]`, as if you were running the commands in a multimode session from the command line.
3. Empty lines and comment lines (starting with the `#` character) are ignored.
4. Each *asadmin* command may contain variable references (e.g. to environment variables or system properties) as described in using-variable-references

NOTE	*asadmin* commands that contain spaces in the value should be

> surrounded with quotations marks and nested quotation marks must be escaped with the `\` character.

Here's a sample of a post boot script used to configure an instance for production uses:

```
#Disable dynamic reloading of applications
set configs.config.server-config.admin-service.das-config.dynamic-reload-enabled
=false

# Set a system property which contains spaces and quotation marks in the property
value
create-system-properties "property.name=This is a \"property\" value"
#Disable auto deployment
set configs.config.server-config.admin-service.das-config.autodeploy-enabled=false

#Optimizing Heap size
delete-jvm-options -client:-Xmx1024:-Xms512
create-jvm-options -server:-Xmx2048:-Xms2048

#Configuring Hazelcast cluster
set-hazelcast-configuration --clusterName=my-production-cluster --target=server-
config

#Using a reference to a system property payara.deployed-app-name
set configs.config.server-config.http-service.virtual-server.server.default-web-
module=${payara.deployed-app-name}

#Using a reference to an environment variable security-enabled
set configs.config.server-config.network-config.protocols.protocol.http-listener-
1.security-enabled=${ENV=security-enabled}
```

Configuring Boot Scripts

When running Payara Micro from the command line, use appropriate command file options to specify the script file locations and the moment in the boot lifecycle when they should be executed:

`--prebootcommandfile`: provides a file of asadmin commands to run before booting the server (only some commands work at this point) **`--postbootcommandfile`**: provides a file of asadmin commands to run after booting the server **`--postdeploycommandfile`**: provides a file of asadmin commands to run after all deployments have completed

For example:

```
java -jar payara-micro.jar --prebootcommandfile prepare-resources.txt
```

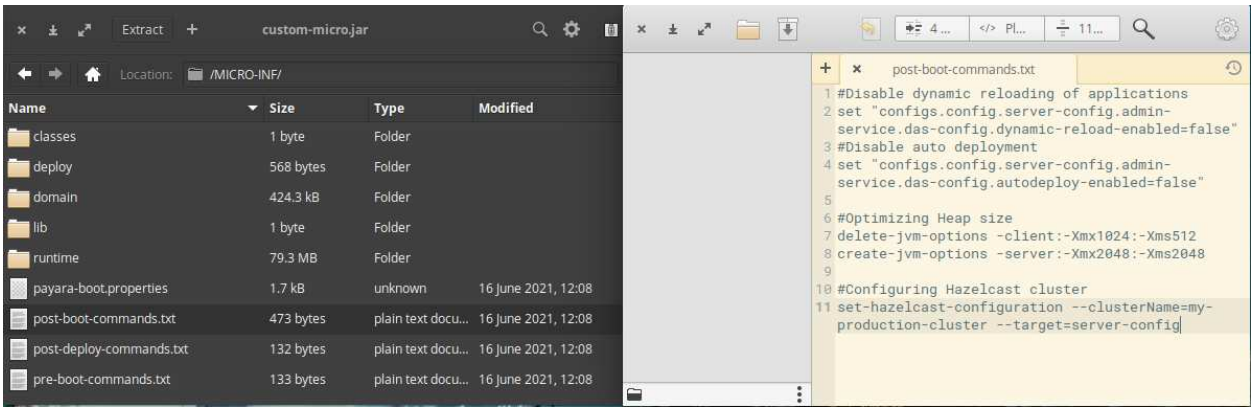
```
--postbootcommandfile ready-production-use.txt --postdeploycommandfile post-
process-apps.txt
```

When creating an Uber JAR using the `--outputUberJAR` option, the scripts are transferred into the `MICRO-INF/` directory of the resulting JAR artifact as the `pre-boot-commands.txt`, `post-boot-commands.txt` and `post-deploy-commands.txt` files.

For example, when generating the Uber JAR like this:

```
java -jar payara-micro.jar --prebootcommandfile prepare-resources.txt
--postbootcommandfile ready-production-use.txt --outputUberJar custom-micro.jar
```

Examining the JAR structure will confirm that the scripts have been transferred correctly:



Boot Scripts with Persistent Root Directory

A command in a boot script usually results in change of configuration in `config/domain.xml`. When persistent root directory is reused between restarts, it may cause some commands to fail, particularly those that create new resources, like for example `create-jdbc-connection-pool`. However, failure of a script command is not critical for server startup, just a warning is logged in such case.

By utilizing "--warmup", these commands can be run as part of preparation of root directory rather than run on every startup. That also spares startup time for productive instance.

When a command uses using-variable-references, the values for variables are determined during execution of the script, before executing the command and changing the configuration. If a variable value should be evaluated each time Payara Micro starts, then executing a script in `--warmup` mode will not suffice and the script needs to be passed as argument also for productive runs.

Restrictions on Pre-Boot Scripts

Keep in mind when writing a **pre-boot** command script, that the commands prepared will be running before the server has booted, which means not all available asadmin subcommands will yield results (for example deploying or listing applications).

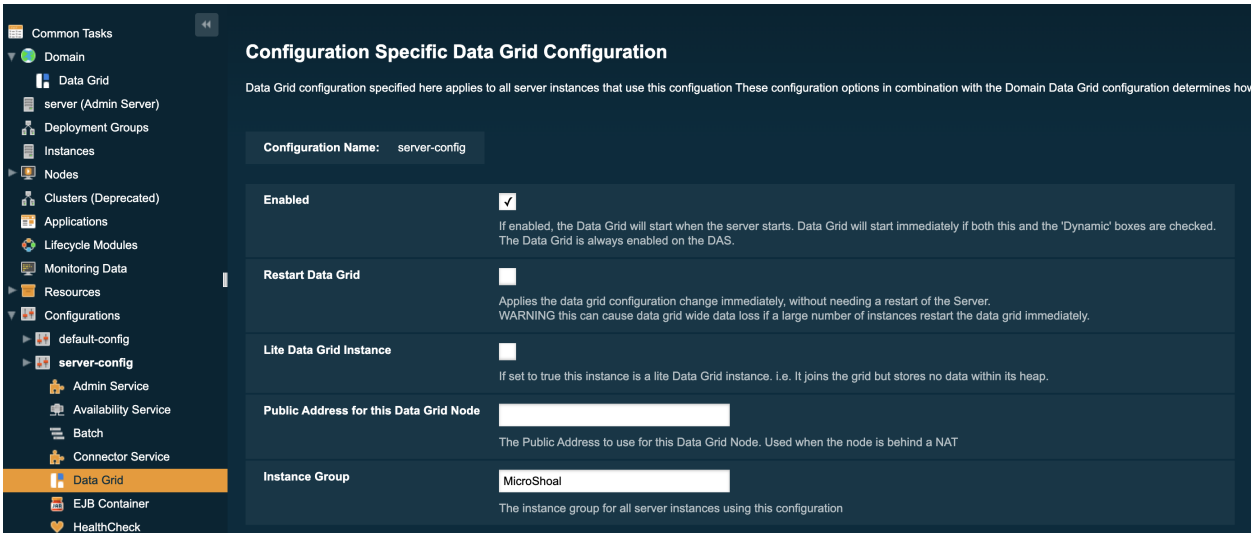
TIP	It's recommended to create *preboot* scripts that use simple set commands to change the configuration of existing resources (adding new resources might not work in all cases) like network listeners, standard domain configuration and such.

Sending Asadmin Commands to Payara Micro from a Domain Administration Server

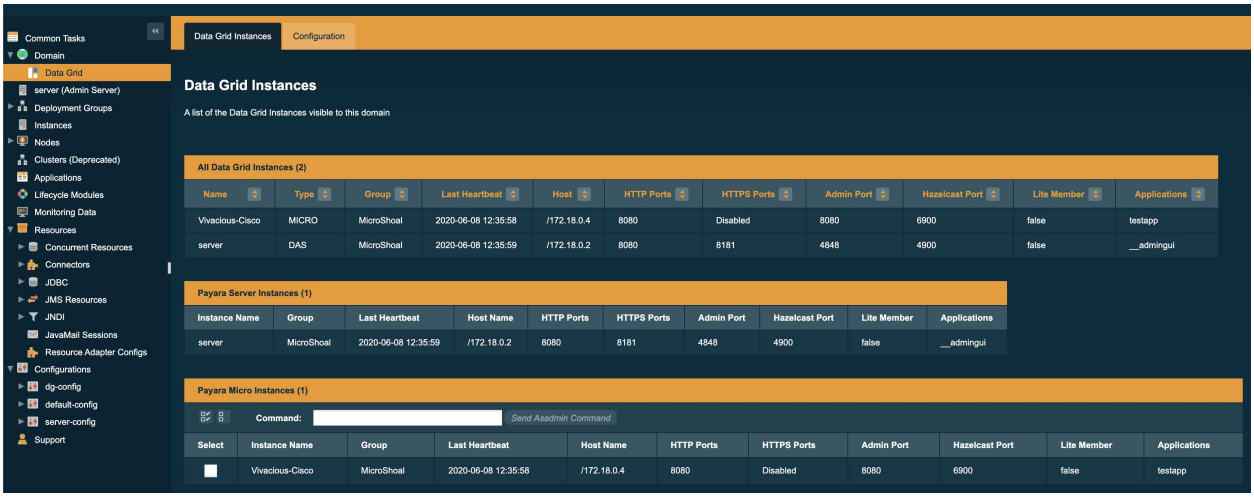
Asadmin commands can be sent from the Domain Administration Server (DAS) to Payara Micro instances that are running within the same data grid as the DAS.

How to Send Asadmin Commands to Payara Micro Using the Admin Console

Sending an Asadmin command is performed through the Domain Data Grid. This Data Grid is powered by Hazelcast and is enabled by default. You can verify if it actived by looking under the *server-config* configuration under the *Data Grid* item.

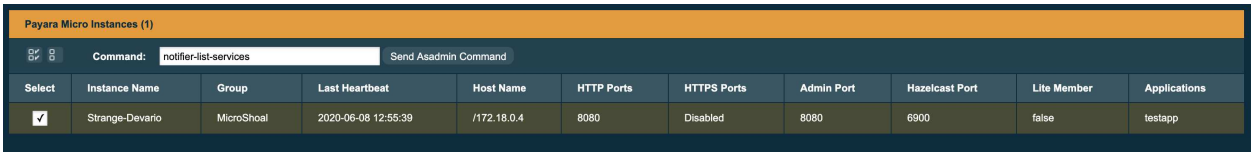


All the members can be seen under the *Data Grid* entry of the Domain item on the left menu. At the bottom of the page, you will find a table filled with the Payara Micro instances:



To send an administration command:

1. Select the recipient nodes
2. Enter the asadmin command in the textbox
3. Press **Send Asadmin Command**



Sending Asadmin Commands from the Command Line

To send administration commands using the command line there is a new `asadmin` command for the Domain Administration Server - `send-asadmin-command`. The asadmin command is entered using the `--command` option, which only accepts a single command as an input (including said commands own arguments).

As an example of sending a command from the command line:

```
asadmin send-asadmin-command --command deploy /home/user/application --contextRoot
=/example
```

Payara Micro CRaC Support

CAUTION	This feature is in Tech Preview, and not recommended for use in production environments.

Payara Enterprise Micro supports creating and restoring from checkpoints using Azul's Zulu JDK Coordinated Restore at Checkpoint (CRaC). Coordinated Restore at Checkpoint (CRaC) is a JDK project that allows you to start Java programs with a shorter time to first transaction, combined with less time and resources to achieve full code speed. CRaC effectively takes a snapshot of the Java process (checkpoint) when it is fully warmed up, then uses that snapshot to launch any number of JVMs from this captured state.

Payara Enterprise Micro specifically supports creating checkpoints during the boot of an instance. Creation and restoration of custom checkpoints during the active runtime of a Payara Enterprise Micro instance is not currently supported.

Requirements

- A CRaC JDK: The standard Zulu JDK provided by Azul does not contain the CRaC extension; you have to download those JDKs specifically marked as CRaC JDKs.
- Linux OS: CRaC only currently works specifically in Linux. Native macOS or Windows are not supported.
- If using the default 'criu' CRaC engine then you will typically need to run Payara Micro

with root permissions due to how the CRIU backend works. It is highly recommended that you use the 'warp' engine instead to avoid this requirement.

Usage

Checkpointing will only occur when using a CRaC JDK and using the `--warmup` option.

As a minimum, you must also specify the `-XX:CRaCCheckpointTo` JVM option to create a checkpoint. It is highly recommended that you use the 'warp' engine by specifying `-XX:CRaCEngine=warp` on both your checkpoint and restore commands.

The `--rootdir` option should also be specified to not unpack Payara Enterprise Micro to a temporary directory which may be cleared away.

```
java -XX:CRaCCheckpointTo=/opt/checkpoint -XX:CRaCEngine=warp
payara-micro.jar --warmup --rootdir /opt/micro myapp.war
```

TIP

Ensure the directories specified by `-XX:CRaCCheckpointTo` and `--rootdir` are both empty to prevent pollution by any previous runs

Payara Enterprise Micro will exit once the checkpoint has been created. To restore from the generated checkpoint, invoke `java` again like so:

```
java -XX:CRaCEngine=warp -XX:CRaCRestoreFrom=/opt/checkpoint
```

TIP

You cannot mix-and-match CRaC engines e.g. checkpoint with 'warp' and restore with 'criu'. Ensure you use the same engine for both; the restoring Java process won't "remember" which engine you used to checkpoint.

The full array of CRaC JVM options can be found in the Azul documentation: CRaC JDK VM Options

Checkpoint Timing

Payara Enterprise Micro supports creating a checkpoint at one of two places during instance initialisation, controlled by the `-Dfish.payara.crac.checkpointAfterDeployment` system property.

When set to `false` (default), a checkpoint will be created after the initial bootstrap and pre-boot configuration. Application deployment, post-boot, and post-deploy configuration will occur after restoration. When set to `true`, a checkpoint will be created after all applications have been deployed and post-boot configuration files have been applied. Post-deploy configuration files will be applied after restoration.

You may only create a checkpoint at one of these timings, not both.

NOTE	While checkpointing after application deployment will speed up the "ready time" of the instance upon restoration, the nature and content of any deployed applications may lead to checkpoint or restoration errors if resources not managed by Payara Enterprise Micro lack CRaC instrumentation.

Docker

CRaC may be utilised in Docker by creating a custom Docker image. The default Payara Enterprise Micro image does not utilise a CRaC JDK and so cannot be used.

See custom-crac-image for details.

Database Management

Logging JDBC Calls in Payara Micro

Payara Micro supports SQL Tracing which brings powerful operational diagnostics to your microservices platform.

Enabling Logging JDBC Calls

To enable SQL Tracing of JDBC on applications deployed on Payara Micro, use can use the annotation or deployment descriptor methods described below.

Application Deployment

In Java EE 7+ applications, a JDBC data source can be deployed by adding the `@DataSourceDefinition` annotation to a managed component (like and `@Stateless` EJB bean for example). The **Log JDBC Calls** setting can be configured using this annotation as well:

```
@DataSourceDefinition(
    name = "java:app/MyApp/MyDS",
    className = "org.h2.jdbcx.JdbcDataSource",
    url = "jdbc:h2:mem:test",
    properties = {"fish.payara.log-jdbc-calls=true"})
```

The Data source definition can also be added to a deployment descriptor of an application, for example in the *web.xml* standard deployment descriptor:

```
<data-source>
   <name>java:global/ExampleDataSource</name>
   <class-name>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</class-name>
   <server-name>localhost</server-name>
   <port-number>3306</port-number>
   <database-name>mysql</database-name>
   <user>test</user>
   <password>test</password>
   <!-- Example of how to use a Payara specific custom connection pool setting -->
   <property>
       <name>fish.payara.log-jdbc-calls</name>
       <value>true</value>
   </property>
</data-source>
```

or in the definition of a `jdbc-connection-pool` in a *payara-resources.xml* file:

```
<jdbc-connection-pool name="examplePool"
        res-type="javax.sql.DataSource"
        datasource-classname="org.h2.jdbcx.JdbcDataSource" log-jdbc-calls="true">
    <property name="user" value="test"/>
    <property name="password" value="test"/>
    <property name="url" value="java:global/ExampleDataSource"/>
</jdbc-connection-pool>
```

This can also be done in the creation of a JDBC Connection Pool with the `--logjdbccalls` option.

```
asadmin create jdbc-connection-pool --datasourceclassname
org.h2.jdbcx.JdbcDataSource --restype javax.sql.XADataSource --logjdbccalls=true
examplePool
```

See Also

- "Slow SQL Logging in Payara Micro"
- "SQL Trace Listeners in Payara Micro"

- sql-management-overview

SQL Trace Listeners in Payara Micro

Payara Micro also supports SQL Trace Listeners which brings powerful operational diagnostics to your microservices platform.

Adding a SQL Trace Listener

To add a SQL Trace Listener to your data source used by applications deployed on Payara Micro, you can use the annotation or deployment descriptor methods described below.

Application Deployment

In Java EE 7+ applications, a JDBC data source can be deployed by adding the `@DataSourceDefinition` annotation to a managed component (like and `@Stateless` EJB bean for example). SQL Trace Listener classes can be configured using this annotation, like in the following example:

```
@DataSourceDefinition(
    name = "java:app/MyApp/MyDS",
    className = "org.h2.jdbcx.JdbcDataSource",
    url = "jdbc:h2:mem:test",
    properties = {"fish.payara.sql-trace-
listeners=fish.payara.examples.payaramicro.datasource.example.CustomSQLTracer"})
```

The Data source definition can also be added to a deployment descriptor of an application, for example in the *web.xml* standard deployment descriptor:

```
<data-source>
    <name>java:global/ExampleDataSource</name>
    <class-name>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</class-name>
    <server-name>localhost</server-name>
    <port-number>3306</port-number>
    <database-name>mysql</database-name>
    <user>test</user>
    <password>test</password>
    <!-- Example of how to use a Payara specific custom connection pool setting -->
    <property>
       <name>fish.payara.sql-trace-listeners</name>
       <value>
fish.payara.examples.payaramicro.datasource.example.CustomSQLTracer</value>
    </property>
</data-source>
```

See Also

- "Slow SQL Logging in Payara Micro"
- "JDBC Call Logging in Payara Micro"
- sql-management-overview

Slow SQL Logging in Payara Micro

Payara Micro supports slow logging of SQL queries which brings powerful operational diagnostics to your microservices platform.

Enabling Slow SQL Logging

To enable it on applications deployed on Payara Micro, you can use the annotation or deployment descriptor methods described below.

Application Deployment

In Java EE 7+ applications, a JDBC data source can be deployed by adding the `@DataSourceDefinition` annotation to a managed component (like and `@Stateless` EJB bean for example). The **Slow Query Log Threshold** time can be configured using this annotation, like in the following example:

```
@DataSourceDefinition(
    name = "java:app/MyApp/MyDS",
    className = "org.h2.jdbcx.JdbcDataSource",
    url = "jdbc:h2:mem:test",
    properties = {"fish.payara.slow-query-threshold-in-seconds=5"})
```

The Data source definition can also be added to a deployment descriptor of an application, for example in the *web.xml* standard deployment descriptor:

```
<data-source>
    <name>java:global/ExampleDataSource</name>
    <class-name>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</class-name>
    <server-name>localhost</server-name>
    <port-number>3306</port-number>
    <database-name>mysql</database-name>
    <user>test</user>
    <password>test</password>
    <!-- Example of how to use a Payara specific custom connection pool setting -->
    <property>
       <name>fish.payara.slow-query-threshold-in-seconds</name>
       <value>5</value>
    </property>
```

```
</data-source>
```

See Also

- "JDBC Call Logging in Payara Micro"
- "SQL Trace Listeners in Payara Micro"
- * sql-management-overview

Logging and Monitoring

Logging

Logging to a File

This section describes how to print all the Payara Micro log messages into a file.

Logging to a File from the Command Line

To print all of the Payara Micro log messages into a file from the command line, you will need to use the `--logToFile` option, followed by either a path to where you want to put the log file or by giving the name of a file you want to print the logs into. If a file name is not given, a default file called *payara-server.log* is generated.

```
java -jar payara-micro.jar --logToFile /home/user/PayaraMicro.log
```

Logging to a File Programmatically

To print all the Payara Micro log messages into a file programmatically, you will need to use `setUserLogFile(String filePath)` method.

```
import fish.payara.micro.PayaraMicro;
import fish.payara.micro.BootstrapException;

public class EmbeddedPayara{
    public static void main(String[] args) throws BootstrapException{
         PayaraMicro.getInstance().setUserLogFile("/home/user/PayaraMicro.log"
).bootStrap();
    }
}
```

Configuration

You can specify a `Logging Properties File` to configure the JUL logger for Payara

Micro.

Options	Type	Description	Default	Mandatory
`--logProperties`	`String`	File with logging configuration		No
`--enableDynamicLogging`	`Boolean`	Enable dynamic reloading of configuration	False	No

Example

You can provide a logging configuration file that uses the standard JUL notation RDP Logging Config

```
java -jar payara-micro.jar --logProperties micro-logging.properties
--enableDynamicLogging --deploy myapplication.war
```

Configuring the Access Log

Payara Micro supports a number of commands to help you configure access logging. It is easiest to configure access logging before it is bootstrapped, however you can edit `domain.xml` also, as Payara Micro can reload that dynamically.

Configuration Option	Description	Default Value
`--accesslog <directory-path>`	Sets user defined directory path for the access log	
`--accesslogformat`	Sets user defined log format for the access log	`%client.name% %auth-user-name% %datetime% %request% %status% %response.length%`
`--accessloginterval`	Sets user defined access log write interval	300
`--accesslogsuffix`	Sets user defined simple date time suffix for access log filenames	`yyyy-MM-dd`
`--accesslogprefix`	Sets user defined prefix for access log filenames	

All options available for configuring Payara Micro can be seen by running the JAR with the `--help` option, or by consulting "Payara Micro Command Line Options"

Usage Examples

Configuration Option	Usage	Result
`--accesslog`	`java -jar payara-micro --accesslog ~/logs/accesslogs`	Access logging is enabled
`--accesslogformat`	`java -jar payara-micro --accesslog ~/logs/accesslogs --accesslogformat %u %t %s %b`	Payara [22/11/2019:09:24:36 00] 200 532
`--accessloginterval`	`java -jar payara-micro --accesslog ~/logs/accesslogs --accessloginterval 0`	Access log write interval is set to write its buffer to file immediately
`--accesslogsuffix`	`java -jar payara-micro --accesslog ~/logs/accesslogs --accesslogsuffix %hh%mm%a`	server_access_log.%09%30 %AM.txt
`--accesslogprefix`	`java -jar payara-micro --accesslog ~/logs/accesslogs --accesslogprefix Production`	Production.2019-11-22.txt

Request Tracing in Payara Micro

The Request Tracing service works the same on Payara Micro and will trace the same events described in its request-tracing.

Usage

Payara Micro exposes a number of command line options to configure request tracing. These options cover the most frequent use cases for Payara Micro, but you can also configure the Request Tracing Service using a **post**-boot command file, giving you access to the asadmin commands and the full range of options they provide.

Configuration Options

Parameter	Summary
`enableRequestTracing`	Enables the service when present. A supplementary parameter can be provided with this option to encapsulate the value and unit options, for example, the following will enable the service with a threshold of `4ms`: `java -jar payara-micro.jar --enableRequestTracing 4ms`
`requestTracingThresholdUnit`	Accepts a String value determining the time unit, defaulting to seconds if no value is given. The accepted strings are (case-insensitive): • `NANOSECONDS` • `MICROSECONDS` • `MILLISECONDS` • `SECONDS` • `MINUTES` • `HOURS` • `DAYS` This is combined with the `requestTracingThresholdValue` setting to determine the threshold time period.
`requestTracingThresholdValue`	Accepts an integer value to determine the threshold value before a request is traced, defaulting to 30 if no value is entered. This is combined with the `requestTracingThresholdUnit` setting to determine the threshold time period.
`enablerequesttracingadaptivesampling`	Enables the *Adaptive Sampling* option when present. When enabled, this allows the sample rate to change to attempt to meet a target count of traces over a specified period of time.

Parameter	Summary
`requesttracingadaptivesamplingtargetcount`	Accepts an integer value to determine the threshold value before a request is traced, defaulting to 6 if no value is entered. Only used if adaptive sampling is enabled.
`requesttracingadaptivesamplingtimevalue`	Accepts an integer value to determine the threshold value before a request is traced, defaulting to 1 if no value is entered. Only used if adaptive sampling is enabled. This is combined with the `requesttracingadaptivesamplingtimeunit` setting to determine the adaptive sampling time period.
`requesttracingadaptivesamplingtimeunit`	Accepts a String value determining the time unit, defaulting to minutes if no value is given. The accepted strings are (case-insensitive): • `NANOSECONDS` • `MICROSECONDS` • `MILLISECONDS` • `SECONDS` • `MINUTES` • `HOURS` • `DAYS` This is combined with the `requesttracingadaptivesamplingtimevalue` setting to determine the adaptive sampling time period. Only used if adaptive sampling is enabled.

Shorthand Notation

To enable and configure request tracing with a single command, the `--enableRequestTracing` option allows you to provide **one** string following it to set the threshold unit and/or value.

In other words, the value and unit need to be a single string with no spaces, so `500ms` would be allowed, but `500 ms` would not. In the second case, the unit is ignored and the

default `SECONDS` is kept.

Notation for units is as follows:

Unit	Acceptable notation
`NANOSECONDS`	• `Nanoseconds` (case-insensitive) • `Nanosecond` (case-insensitive) • `ns`
`MICROSECONDS`	• `Microseconds` (case-insensitive) • `Microsecond` (case-insensitive) • `us` • `¬s`
`MILLISECONDS`	• `Milliseconds` (case-insensitive) • `Millisecond` (case-insensitive) • `ms`
`SECONDS`	• `Seconds` (case-insensitive) • `Second` (case-insensitive) • `s`
`MINUTES`	• `Minutes` (case-insensitive) • `Minute` (case-insensitive) • `Mins` (case-insensitive) • `Min` (case-insensitive) • `m`
`HOURS`	• `Hours` (case-insensitive) • `Hour` (case-insensitive) • `h`

Unit	Acceptable notation
`DAYS`	• `Days` (case-insensitive) • `Day` (case-insensitive) • `d`

Example Usages

java -jar payara-micro.jar --enableRequestTracing: Enable request tracing ****java -jar payara-micro.jar --enableRequestTracing ns***: Enable request tracing and set threshold unit to nanoseconds (***value*** *defaults to* ***30***) ****java -jar payara-micro.jar --enableRequestTracing 2***: Enable request tracing and set threshold value to 2 (***unit*** *defaults to* ***SECONDS***) ****java -jar payara-micro.jar --enableRequestTracing 4ms***: Enable request tracing and set threshold to 4 microseconds

WARNING

Exercise caution when using the shorthand notation together with other parameters. Due to the way the parser works, the last parameter entered will override the previous. For example:

```
java -jar payara-micro.jar --requestTracingThresholdUnit DAYS
--enableRequestTracing 1ns
```

Will enable request tracing, and set the threshold to `1 NANOSECOND`, while

```
java -jar payara-micro.jar --enableRequestTracing 1ns
--requestTracingThresholdUnit DAYS
```

Will enable request tracing, and set the threshold to `1 Day`

API

Jcache in Payara Micro

Payara Micro provides JCache functionality and the documentation for using "JCache in Payara Server" also applies to it.

Payara Micro API

Payara Micro API

This section details documentation on the Payara Micro API. The Javadoc for the most recent version of the Payara Micro API can be found here.

`PayaraMicro.class` methods are used during the bootstrap process, whereas `PayaraMicroRuntime.class` methods are used on running instances.

Configuration Methods

Configuration Methods of `PayaraMicro.class` are detailed here.

Configuration Operand	Description	Get Method	Set Method	Default Value	Command Line Equivalent
Alternate Domain XML	Gets or sets the domain.xml used to override the default server configuration.	`File getAlternateDomainXML()`	`PayaraMicro setAlternateDomainXML(File alternateDomainXML)`	`null`	`--domainConfig`

Configuration Operand	Description	Get Method	Set Method	Default Value	Command Line Equivalent
Cluster Multicast Group	Gets or sets the cluster multicast group for the instance.	`String getClusterMulticastGroup()`	`PayaraMicro setClusterMulticastGroup(String hzMulticastGroup)`	`null` (Default value of 224.2.2.4 set in default domain.xml is not read into instance variable)	`--mcAddress`

Configuration Operand	Description	Get Method	Set Method	Default Value	Command Line Equivalent
Cluster Port	Gets or sets the multicast cluster port.	`int getClusterPort()`	`PayaraMicro setClusterPort(int hzPort)`	-2147483648 (MIN_VALUE) (Default value of 2904 set in default domain.xml is not read into instance variable)	`--mcPort`

Configuration Operand	Description	Get Method	Set Method	Default Value	Command Line Equivalent
Cluster Start Port	Gets or sets the start port number for the Payara Micro instance to listen on for cluster communications.	`int getClusterStartPort()`	`PayaraMicro setClusterStartPort(int hzStartPort)`	-2147483648 (MIN_VALUE) (Default value of 5900 set in default configuration files is not read into instance variable)	`--startPort`
Deployment Directory	Gets or sets the directory to be scanned for archives to deploy.	`File getDeploymentDir()`	`PayaraMicro setDeploymentDir(File deploymentRoot)`	`null`	`--deploymentDir`

Configuration Operand	Description	Get Method	Set Method	Default Value	Command Line Equivalent
HTTP Port	Gets or sets the HTTP port for the instance to bind to.	`int getHttpPort()`	`PayaraMicro setHttpPort(int httpPort)`	-2147483648 (MIN_VALUE) (Default value of 8080 set in default domain.xml is not read into instance variable)	`--port`
Instance Name	Gets or sets the name of the instance.	`String getInstanceName()`	`PayaraMicro setInstanceName(String instanceName)`	Generated Universally Unique Identifier.	`--name`

Configuration Operand	Description	Get Method	Set Method	Default Value	Command Line Equivalent
Maximum HTTP Threads	Gets or sets the maximum number of threads in the HTTP thread pool.	`int getMaxHttpThreads()`	`PayaraMicro setMaxHttpThreads(int maxHttpThreads)`	-2147483648 (MIN_VALUE) (Default value of 10 set in default domain.xml is not read into instance variable)	`--maxHttpThreads`

Configuration Operand	Description	Get Method	Set Method	Default Value	Command Line Equivalent
Minimum HTTP Threads	Gets or sets the minimum number of threads in the HTTP thread pool.	`int getMinHttpThreads()`	`PayaraMicro setMinHttpThreads(int minHttpThreads)`	-2147483648 (MIN_VALUE) (Default value of 10 set in default domain.xml is not read into instance variable)	`--minHttpThreads`
Root Directory	Gets or sets the root configuration directory.	`File getRootDir()`	`PayaraMicro setRootDir(File rootDir)`	`null`	`--rootDir`

Configuration Operand	Description	Get Method	Set Method	Default Value	Command Line Equivalent
HTTPS Port	Gets or sets the HTTPS port for the instance to bind to. An HTTPS port is not bound unless this value is manually set.	`int getSslPort()`	`PayaraMicro setSslPort(int sslPort)`	-2147483648 (MIN_VALUE) (Default value of 8443 set in default domain.xml is not read into instance variable)	`--sslPort`
No Clustering	Gets or sets whether clustering is enabled or disabled for an instance.	`boolean isNoCluster()`	`PayaraMicro setNoCluster(boolean noCluster)`	*false*	`--noCluster`
HTTP Auto-Binding	Enables or Disables auto-binding of the HTTP port for an instance.	`boolean getHttpAutoBind()`	`PayaraMicro setHttpAutoBind(boolean httpAutoBind)`	*false*	`--autoBindHttp`
HTTPS Auto-Binding	Enables or Disables auto-binding of the HTTPS port for an instance.	`boolean getSslAutoBind()`	`PayaraMicro setSslAutoBind(boolean sslAutoBind)`	*false*	`--autoBindSsl`

Configuration Operand	Description	Get Method	Set Method	Default Value	Command Line Equivalent
Auto-Bind Range	Sets the range for HTTP and HTTPS port auto-binding.	`int getAutoBindRange()`	`PayaraMicro setAutoBindRange(int autoBindRange)`	5	`--autoBindRange`

Operation Methods

The other methods of the Payara Micro API that operate Payara Micro instances are detailed here.

The list of methods in the API:

- Deployment Methods
- Setup and Shutdown Methods
- CDI Methods
- Run Methods
- Get Methods
- Library Methods

Deployment Methods

Deployment methods are used for the deployment of applications to Payara Micro instances.

`PayaraMicro.class` Methods

Method	Description	Command Line Equivalent
`PayaraMicro addDeployment(String pathToWar)`	Adds the file found at the location of the `pathToWar` parameter to the list of files to be deployed upon starting the instance.	`--deploy`
`PayaraMicro addDeploymentFile(File file)`	Adds the file associated with the `file` parameter to the list of files to be deployed upon starting the instance.	`--deploy`

`PayaraMicroRuntime.class` Methods

Method	Description
`boolean deploy(String name, InputStream is)`	Deploys an application from an `InputStream` with the name specified.
`boolean deploy(String name, String contextRoot, InputStream is)`	Deploys an application from an `InputStream`, with the specified name and context root.
`boolean deploy(File war)`	Deploys the application located at the *war* parameter path.
`boolean undeploy(String name)`	Un-deploys the specified application.

Setup and Shutdown Methods

These methods are required for setting up and cleaning Payara Micro instances.

`PayaraMicro.class` Methods

Method	Description
`PayaraMicroRuntime bootStrap() throws BootstrapException`	Checks the supplied configuration parameters and starts a Payara Micro instance.
`PayaraMicro getInstance()`	Obtains the static singleton instance of the Payara Micro server. If one does not exist, one will be created and returned.

Method	Description
`PayaraMicro getInstance(boolean create)`	Obtains the static singleton instance of the Payara Micro server. If one does not exist and `create` is set to true, one will be created and returned, otherwise returns *null*.
`void shutdown() throws BootstrapException`	Stops and shuts down the Payara Micro instance.

`PayaraMicroRuntime.class` Methods

Method	Description
`void shutdown() throws BootstrapException`	Stops and shuts down the Payara Micro instance.

CDI Methods

These methods are used for firing CDI Events across running instances.

Method	Description
`void addCDIEventListener(CDIEventListener listener)`	Adds a CDI Event Listener.
`void removeCDIEventListener(CDIEventListener listener)`	Removes a CDI Event Listener.
`void addClusterListener(PayaraClusterListener listener)`	Adds a Cluster Listener.
`void removeClusterListener(PayaraClusterListener listener)`	Removes a Cluster Listener.
`void publishCDIEvent(PayaraClusteredCDIEvent event)`	Publishes a CDI Event.

Run Methods

These methods are used for running `asadmin` commands or `Callable` objects on bootstrapped instances.

Method	Description
`Map<InstanceDescriptor, Future<ClusterCommandResult>> run (String command, String… args )`	Runs an asadmin command on all members of a Payara Micro Cluster.
`Map<InstanceDescriptor, Future<ClusterCommandResult>> run (Collection<InstanceDescriptor> members, String command, String… args )`	Runs an asadmin command on specified members of a Payara Micro Cluster.
`<T extends Serializable> Map<InstanceDescriptor, Future<T>> run (Callable<T> callable)`	Runs a Callable object on all members of a Payara Micro Cluster.
`<T extends Serializable> Map<InstanceDescriptor, Future<T>> run (Collection<InstanceDescriptor> members, Callable<T> callable)`	Runs a Callable object on specified members of a Payara Micro Cluster.

Get Methods

These methods are used for getting information on running Payara Micro instances. For information on the `Get` methods of a non-bootstrapped instance, see the Configuration Methods section.

Method	Description
`Collection<InstanceDescriptor> getClusteredPayaras()`	Returns a collection of instance descriptors for all the Payara Micros instances in a cluster.
`Collection<String> getDeployedApplicationNames()`	Returns the names of the deployed applications.
`String getInstanceName()`	Returns the instance name.
`InstanceDescriptor getLocalDescriptor()`	Returns the instance descriptor of the instance the method is run on.

Library Methods

These methods are used for configuring additional libraries to the instance's classpath.

Method	Description
`void addLibrary(File)`	Adds a new library source (a JAR file or a directory with multiple JAR files) to be considered for the instance's classpath

Running Asadmin Commands on Bootstrapped Instances Using the API

It is possible to execute administration commands programmatically using the "Payara Micro API", using the `PayaraMicroRuntime.run` method.

There are two overloaded forms that can be used:

`*run(String command, String… args )`*: This method executes the specified *asadmin* command on all instances in the runtime's cluster. It returns a `Map<InstanceDescriptor, Future<ClusterCommandResult>>`, which can be used to inspect the results of the execution on each instance the command was executed.

`*run(Collection<InstanceDescriptor> members, String command, String… args )`*: This method executes the specified *asadmin* command on all instances contained in the `Collection` supplied. As with the previous method, it returns a `Map<InstanceDescriptor, Future<ClusterCommandResult>>`, which can be used to inspect the results of the command execution.

> **NOTE** For both methods, it's recommended to use the `getClusteredPayaras()` method exposed by the `PayaraMicroRuntime` class to retrieve a list of all the instances in the cluster and determine which command executions should be filtered.

Here's an example of how to use the first form of the method that executes the `deploy` subcommand to deploy an application and parses the results to check out if the application was successfully deployed on all instances of the cluster:

```
import fish.payara.micro.BootstrapException;
import fish.payara.micro.ClusterCommandResult;
import fish.payara.micro.PayaraMicro;
```

```
import fish.payara.micro.PayaraMicroRuntime;
import fish.payara.micro.data.InstanceDescriptor;
import java.util.Map;
import java.util.concurrent.Future;
import java.util.logging.Logger;

public class EmbeddedPayara {

    private static final Logger LOG = Logger.getLogger(EmbeddedPayara .class
.getName());

    public static void main(String[] args) throws BootstrapException {

        PayaraMicro instance = PayaraMicro.getInstance();
        PayaraMicroRuntime runtime = instance.bootStrap();

        Map<InstanceDescriptor, Future<? extends ClusterCommandResult>> results =
runtime.run("deploy", "application.war");
        for(InstanceDescriptor descriptor : results.keySet()){
            ClusterCommandResult commandResult = (ClusterCommandResult)results.get
(descriptor);
            if(commandResult.getExitStatus() == ClusterCommandResult.ExitStatus
.SUCCESS){
                LOG.info("Application was successfully deployed on cluster instance
" + descriptor.getInstanceName());
            }
        }
    }
}
```

Extensions

JCA Support in Payara Micro

Payara Micro supports the deployment of Java EE Connector Architecture (JCA) adapters which can enable communication with external services not natively supported by Payara Micro.

These can include JMS brokers, cloud messaging providers like Amazon SQS or existing legacy systems. In the case of JMS, for example, Payara Micro can be used for both inbound and outbound messaging to a JMS broker using a standard Message Driven Bean (MDB) and can therefore take advantage of pooling behaviour available to all types of EJB.

Installing a JCA Adapter

A JCA Resource Archive (RAR) is installed by deploying it to Payara Micro in the same way as an application archive (e.g. a WAR) is deployed. The following shows an example using an ActiveMQ (JMS 1.1) RAR:

```
java -jar payara-micro.jar --deploy activemq-rar-5.14.5.rar
```

The connector is subsequently available to all other applications deployed to Payara Micro. An application using a connector and the connector itself can be deployed simultaneously:

```
java -jar payara-micro.jar --deploy activemq-rar-5.14.5.rar --deploy myapp.war
```

NOTE	When deploying multiple files to Payara Micro, the order of `--deploy` options is respected so, in the above example, the ActiveMQ RAR will be deployed before the `myapp.war` application.

See Also

- "User Guide: Using Payara Micro as a JMS Client"

Persistent EJB Timers

Payara Micro supports persistent EJB timers which are stored across Payara Micro instances within the cluster, backed by distributed Hazelcast cache.

The timers are persisted as long as the cluster is running, even when the member that created the timer goes down.

In order to make use of a persistent timer in Payara Micro, an application must declare a timer to be persistent, as defined by the EJB specification. This is actually the default behavior, unless timers are specified as **non-persistent**.

The second requirement is to specify a name for every Payara Micro instance which runs an application with persistent timers. Each such instance has to have its own unique name.

CAUTION	Persistent timers are **NOT** coordinated across a Payara Micro cluster. They are always executed on an instance with the same name that created the timers.

Requirements to Setup a Persistent Timer

1. The timer has to be persistent, i.e. the value of `persistent` attribute is not forcibly set to `false`
2. A Payara Micro instance is given a unique name using `--name` argument
3. A Payara Micro instance is a member of a Hazelcast cluster

NOTE	The Hazelcast cluster can be formed by other Payara Micro instances, Payara Server instances with Hazelcast turned on, or by standalone Hazelcast instances.

How EJB Timers Are Persisted

Once timers are scheduled, either by the `@Schedule` annotation or using the `TimerService`, the schedule is stored into the Hazelcast distributed cache. The schedule is given a key, which is the name of the Payara Micro instance that created the timer. If that instance goes down, the timer will be recreated on another instance with the same name once it joins the cluster. Until that time, the timer becomes inactive.

The timer would not be recreated on other members of the cluster or any new members of the cluster with a different name.

Remote CDI Events in Payara Micro

Payara Micro can listen to and fire CDI events across the cluster of Payara Micro instances, and also across Payara Server instances within the same cluster. Payara Micro supports the same functionality for "remote CDI events" as provided by Payara Server.

Payara Micro also provides additional methods to fire events and register listeners programmatically.

These methods are used for firing CDI Events across running instances.

Method	Description
`void addCDIEventListener(CDIEventListener listener)`	Adds a CDI Event Listener.
`void removeCDIEventListener(CDIEventListener listener)`	Removes a CDI Event Listener.
`void addClusterListener(PayaraClusterListener listener)`	Adds a Cluster Listener.
`void removeClusterListener(PayaraClusterListener listener)`	Removes a Cluster Listener.
`void publishCDIEvent(PayaraClusteredCDIEvent event)`	Publishes a CDI Event.

NOTE This feature has been available in Payara Micro long before it was available in Payara Server.

Manual Initialization of the Event Bus

On prior releases, it was necessary to initialize the event bus using `ClusteredCDIEventBus.initialize()` manually upon startup. Although this method is still available in the API, it is not necessary anymore, since the event bus is automatically initialized after an application is started.

Running Callable Objects on Bootstrapped Instances

Like with running asadmin commands, there are two methods available for running `Callable` objects, both also named `run`.

The two methods also work in a similar way to the two asadmin run methods:

- **`run(Callable<T> callable)`**: runs the specified Callable on all instances in a runtime's cluster.
- **`run(Collection<InstanceDescriptor>`**: members, Callable<T> callable)` runs the Callable on a subset of the instances in a runtime's cluster.

Both methods return a `Map` object with a key/value type of `<InstanceDescriptor, Future<T>>`, where the type variable *T* is Serializable.

Payara Micro Docker Image Overview

Overview of the usage and details of the official Payara Micro Enterprise Docker image.

Quick Start

Launch a new Payara Micro instance running in a container listening in port `8080` with the following command:

```
docker run -p 8080:8080 nexus.payara.fish:5000/payara/micro
```

Keep in mind the following:

- The instance will start with no deployed applications.
- Accessing the default HTTP endpoint will yield a `404 - Not Found` response.
- The instance's JVM process runs under the `payara` user.

IMPORTANT	It is recommended to specify the tag name of the Docker image that you plan to use, to guarantee that the correct Payara Enterprise version is used on your environment for guaranteed long-term stability

Using a specific version

```
docker run -p 8080:8080 nexus.payara.fish:5000/payara/micro:7.2.0
```

AMD64 & ARM64 compatible images are provided. The Docker client should automatically download the appropriate image for your architecture.

Exposed Ports

The default ports that are exposed by this image are:

- 8080 - HTTP listener

- 6900 - Datagrid's Hazelcast port

Installation Details

The Payara Micro JAR file is located in the `/opt/payara/` directory (identified by the `HOME_DIR` environment variable), which also is the default work directory of the image.

IMPORTANT	The directory name is deliberately free of any versioning so that any scripts written to work with one version can be seamlessly migrated to the latest docker image.

Application Deployment

Payara Micro will deploy applications at startup time, depending on how you prefer to instruct the instance to execute the deployments. However, you have to make sure that the container has access to your application artefacts.

You can instruct the container to deploy applications using the following mechanisms:

- Mount a deployment directory (local or in the network) as a Docker volume
- Prepare a custom Docker image that includes your application artefacts.
- Deploy your applications from a remote Maven repository.

Deploy Applications from a Directory

You can instruct the container to deploy all application artefacts that belong to a directory that is mounted as Docker volume mapped to the `/opt/payara/deployments` (which is defined as the value of the `$DEPLOY_DIR` environment variable) local directory.

The following command will start a new Payara Micro instance container and deploy all artefacts located in the `~/payara-micro/applications` directory:

```
docker run -p 8080:8080 -v ~/payara-micro/applications:/opt/payara/deployments
nexus.payara.fish:5000/payara/micro:7.2.0
```

To only deploy a specific artefact within the directory, simply pass the `--deploy`

argument to the container's run command and specify which application to deploy:

```
docker run -p 8080:8080 -v ~/payara-micro/applications:/opt/payara/deployments
nexus.payara.fish:5000/payara/micro:7.2.0 --deploy
/opt/payara/deployments/myapplication.war
```

NOTE Make sure to prefix the artefact names with the absolute path to the local deployment directory.

Deploy Applications Using a Custom Image

You can also prepare a custom image based on the official Payara Micro Docker image and copy all application artefacts to the `$DEPLOY_DIR` directory. This way, you can re-package your applications in a simple distributable unit.

The following sample *Dockerfile* can be used to prepare a custom Payara Micro image that deploys the `myapplication.war` artefact at runtime:

```
FROM nexus.payara.fish:5000/payara/micro:7.2.0
COPY myapplication.war $DEPLOY_DIR
```

Create Docker Images Utilising CRaC

CAUTION This feature is in Tech Preview, and not recommended for use in production environments.

Below is an example of creating a custom image to use Payara Micro with a CRaC JDK. This dockerfile makes use of your own Payara Micro installation and will create the checkpoint as part of the image. Containers created from the image will restore the application from the checkpoint.

```
FROM azul/zulu-openjdk:21-jdk-crac-latest

EXPOSE 6900 8080

ENV HOME_DIR=/opt/payara
ENV PAYARA_DIR=${HOME_DIR}/micro \
    DEPLOY_DIR=${HOME_DIR}/deployments \
    CHECKPOINT_DIR=${HOME_DIR}/checkpoint
ENV JVM_ARGS="-Djdk.util.zip.disableZip64ExtraFieldValidation=true \
    -XX:+UseContainerSupport \
    -XX:MaxRAMPercentage=70.0 \
```

```
    -Xss512k \
    -XX:CRaCCheckpointTo=${CHECKPOINT_DIR} \
    -XX:CRaCEngine=warp"

RUN true \
    && mkdir -p "${HOME_DIR}" \
    && addgroup --gid 1000 payara \
    && adduser --system --uid 1000 --no-create-home --shell /bin/bash --home "
${HOME_DIR}" --gecos "" --ingroup payara payara \
    && echo payara:payara | chpasswd \
    && mkdir -p "${PAYARA_DIR}" \
    && mkdir -p "${DEPLOY_DIR}" \
    && mkdir -p "${CHECKPOINT_DIR}" \
    && chown -R payara:payara ${HOME_DIR} \
    && true

USER payara
WORKDIR ${HOME_DIR}

COPY --chown=payara:payara payara-micro.jar .
COPY --chown=payara:payara myapplication.war $DEPLOY_DIR

RUN java \
    ${JVM_ARGS} \
    -Dfish.payara.crac.checkpointAfterDeployment=true \
    -jar payara-micro.jar \
    --deploydir $DEPLOY_DIR \
    --rootdir $PAYARA_DIR \
    --warmup || true

SHELL ["/bin/bash", "-c"]
ENTRYPOINT java -XX:CRaCEngine=warp -XX:CRaCRestoreFrom=${CHECKPOINT_DIR}
```

To build and run this docker image, the following commands can be used.

```
docker build . -t payara-enterprise-micro-crac-example
docker run -d -p 8080:8080 -p 6900:6900 --name payara-enterprise-micro-crac-
example-1 payara-enterprise-micro-crac-example
```

Deploy Applications from a Maven Repository

If your application's artefact is already available in a Maven repository, you can instruct the container to download and deploy it directly. To do this, you'll have to pass the `--deployFromGAV` argument with the GAV coordinates of your artefact to the run command used to launch the container.

Here's an example of how to deploy a sample application located in a local Maven repository:

```
docker run -p 8080:8080 nexus.payara.fish:5000/payara/micro:7.2.0 --deployFromGAV
```

```
"fish.payara.examples:my-application:1.0-SNAPSHOT" --additionalRepository
https://172.17.0.10/content/repositories/snapshots
```

Notice that the example also uses the `--additionalRepository` argument to specify the URL of the repository where the artefact lives since by default, Payara Micro will only look for artefacts in the Maven Central repository.

Configuration

Context Root Configuration

By default, the context root of each deployed application corresponds to the artefact's file name. You can override this setting by using the `--contextroot` startup argument.

> **NOTE** To deploy an application using the default context root (`/`), set its value to `ROOT`

Here are some basic examples:

One application deployed, uses the `myRoot` context root

```
docker run -p 8080:8080 -v ~/payara-micro/applications:/opt/payara/deployments
nexus.payara.fish:5000/payara/micro:7.2.0 --deploymentDir /opt/payara/deployments
--contextroot myRoot
```

First application in the directory uses the `/` context root

```
docker run -p 8080:8080 \
 -v ~/payara-micro/applications:/opt/payara/deployments
nexus.payara.fish:5000/payara/micro:7.2.0 --deploy
/opt/payara/deployments/myapplication.war --contextroot ROOT
```

You can also prepare a custom Docker image that overrides the default `CMD` instruction to specify the context root like this:

```
FROM nexus.payara.fish:5000/payara/micro:7.2.0
COPY myapplication.war $DEPLOY_DIR
CMD ["--deploymentDir", "/opt/payara/deployments", "--contextroot", "my"]
```

Disable the Data Grid

Payara Micro will start in "clustering" mode by booting up the "Data Grid", allowing other instances reachable in the network to join the grid automatically.

The Data Grid initialization and maintenance consumes extra resources, so in cases where clustering is not needed, it is recommended to disable the Data Grid completely.

To disable the Data Grid, you can pass the `--noHazelcast` argument to the entry point of the run command:

```
docker run -p 8080:8080 -v ~/payara-micro/applications:/opt/payara/deployments
nexus.payara.fish:5000/payara/micro:7.2.0 --noHazelcast
```

Disable Clustering

By default, Payara Micro will start with hazelcast enabled, allowing other instances reachable in the network to join the datagrid automatically and cluster.

Disabling Hazelcast with the --noHazelcast option will also disable all features that depend on Hazelcast, including JCache. The `--noCluster` option allows you to keep Hazelcast and therefore all features depending on Hazelcast, but disable clustering. This will significantly improve performance and is the recommended option if you require Hazelcast dependant features, but do not intend to use clustering.

To disable clustering, you can pass the `--noCluster` argument to the entry point of the run command:

```
docker run -p 8080:8080 -v ~/payara-micro/applications:/opt/payara/deployments
nexus.payara.fish:5000/payara/micro:7.2.0 --noCluster
```

Using Environment Variables

The following environment variables can be used to configure multiple settings of the Payara Micro instance. They can be either specified in a custom image's `Dockerfile` or passed to the `docker run` command via the `--env` or `--env-file` arguments:

Name	Description	Default Value
`MEM_MAX_RAM_PERCENTAGE`	Value for the JVM argument `-XX:MaxRAMPercentage` which indicates the percentage of memory assigned to the container that can be used by the Java process	`70`
`MEM_XSS`	Value for the JVM argument `-Xss` which controls the stack size	`512K`
`JVM_ARGS`	Additional JVM arguments which will be used to configure the Payara Servers DAS JVM settings	`-Djdk.util.zip.disableZip64ExtraFieldValidation=true`

The following is a list of variables used by the Docker image to set up the Payara Micro instance startup, so it is not recommended to alter their values:

Variable name	Description	Value
`HOME_DIR`	The directory containing the Payara Micro JAR binary and the scripts used to run the instance.	`/opt/payara`

Read Only Support Considerations

The Payara Micro Docker image requires write access to the `HOME_DIR`, and `/tmp` directories to function. If you wish to use `--read-only` Docker containers (or the equivalent `--readOnlyRootFilesystem` property in Kubernetes), you must mount volumes at these directories which the `payara` user has write access to. This can be done using anonymous or tmpfs volumes if you do not wish to persist the data outside the life of the container. Named volumes or bind mounts may be used if you wish the data to persist outside the life of the container.

Here is an example `run` command for a read-only Docker container which will delete itself and its anonymous volumes when stopped, deploying all artefacts located in the `~/payara-micro/applications` directory:

```
docker run -p 8080:8080 --read-only --rm -v /opt/payara -v /tmp -v ~/payara-
micro/applications:/opt/payara/deployments
nexus.payara.fish:5000/payara/micro:7.2.0
```

Payara Embedded Documentation

Overview

This guide explains how to run applications in embedded Payara Server and to develop applications in which Payara Server is embedded.

Payara Server Embedded Server Guide

This document explains how to run applications in embedded Payara Server and to develop applications in which Payara Server is embedded.

This document is for software developers who are developing applications to run in an embedded instance of Payara Server.

Introduction to Embedded Payara Server

Embedded Payara Server enables you to use Payara Server as a library. Embedded Payara Server also enables you to run Payara Server inside any Virtual Machine for the Java platform (Java Virtual Machine or JVM).

No installation or configuration of embedded Payara Server is required. The ability to run Payara Server inside applications without installation or configuration simplifies the process of bundling Payara Server with applications.

You can use embedded Payara Server in the following ways:

- With the Embedded Server API (see "Including the Payara Server Embedded Server API in Applications")
- With the EJB Embeddable API (see "Using the EJB Embeddable API with Embedded Payara Server"

Embedded Payara Server File System

The following Embedded Payara Server directories and files are important if you are referencing a non-embedded installation of Payara Server:

- "Installation Root Directory"

- "Instance Root Directory"
- "The `domain.xml` File"

Installation Root Directory

The installation root directory, represented as `as-install`, is the parent of the directory that embedded Payara Server uses for configuration files. This directory corresponds to the base directory for an installation of Payara Server. Configuration files are contained in the following directories in the base directory for an installation of Payara Server:

- `domains`
- `lib`

Instance Root Directory

The instance root directory, represented as `domain-dir`, is the parent directory of a server instance directory. Embedded Payara Server uses the server instance directory for domain configuration files.

NOTE

If you have valid non-embedded Payara Server `as-install` and `domain-dir` directories, specify both in the `BootstrapProperties` and `GlassFishProperties` classes respectively as described in "Creating and Configuring an Embedded Payara Server".

If domain-dir is not specified, Payara Server creates a directory named `gfembed random-number tmp` in a temporary directory, where random-number is a randomly generated 19-digit number. Payara Server then copies configuration files into this directory. The temporary directory is the value of the system property `java.io.tmpdir`. You can override this value by specifying the `glassfish.embedded.tmpdir` property in the `GlassFishProperties` class or as a system property.

The `domain.xml` File

Using an existing `domain.xml` file avoids the need to configure embedded Payara

Server programmatically in your application. Your application obtains domain configuration data from an existing `domain.xml` file. You can create this file by using the administrative interfaces of an existing Payara Server installation.

To specify an existing `domain.xml` file, invoke the `setConfigFileURI` method of the `GlassFishProperties` class as described in "Creating and Configuring an Embedded Payara Server".

Including the Payara Server Embedded Server API in Applications

Payara Server provides an application programming interface (API) for developing applications in which Payara Server is embedded.

Setting the Class Path

Payara Embedded has two distribution variants, `payara-embedded-all` and `payara-embedded-web`, functionally these are the same as their standalone counterparts, Payara Server Full Profile and Payara Server Web Profile respectively.

To enable your applications to locate the class libraries for embedded Payara Server, add one of the following JAR files to your class path:

`payara-embedded-all.jar`: Contains classes needed for deploying all Jakarta EE application types. Can be setup as a Maven dependency with the following GAV coordinates:

+

```
<dependency>
    <groupId>fish.payara.extras</groupId>
    <artifactId>payara-embedded-all</artifactId>
    <version>7.2.0</version>
    <type>jar</type>
</dependency>
```

- **`payara-embedded-web.jar`**: Contains classes needed for deploying strictly Jakarta EE web applications. Can be setup as a Maven dependency with the following GAV coordinates:

```
<dependency>
    <groupId>fish.payara.extras</groupId>
    <artifactId>payara-embedded-web</artifactId>
    <version>7.2.0</version>
    <type>jar</type>
</dependency>
```

In addition, add to the class path any other JAR files or classes upon which your applications depend.

For example, if an application uses a database other than H2, include the JDBC driver JAR files in the class path accordingly.

Creating, Starting, and Stopping Embedded Payara Server

Before you can run applications, you must set up and run the embedded Payara Server.

Creating and Configuring an Embedded Payara Server

To create and configure an embedded Payara Server, perform these tasks:

1. Instantiate the `org.glassfish.embeddable.BootstrapProperties` class.
2. Invoke any methods for configuration settings that you require. This is optional.
3. Invoke the `GlassFishRuntime.bootstrap()` or `GlassFishRuntime.bootstrap(BootstrapProperties)` method to create a `GlassFishRuntime` object.
4. Instantiate the `org.glassfish.embeddable.GlassFishProperties` class.
5. Invoke any methods for configuration settings that you require. This is optional.
6. Invoke the `glassfishRuntime.newGlassFish(GlassFishProperties)` method to create a `GlassFish` object.

The methods of the `BootstrapProperties` class for setting the server configuration are listed in the following table. The default value of each configuration setting is also listed.

Table 177. Methods of the `BootstrapProperties` Class

Purpose	Method	Default Value
References an existing "Installation Root Directory", also called `as-install`	`setInstallRoot(String as-install)`	None.

The methods of the `GlassFishProperties` class for setting the server configuration are listed in the following table. The default value of each configuration setting is also listed.

Table 178. Methods of the `GlassFishProperties` Class

Purpose	Method	Default Value
References an existing "Instance Root Directory", also called `domain-dir`	`setInstanceRoot(String domain-dir)`	In order of precedence: • `glassfish.embedded.tmpdir` property value specified in `GlassFishProperties` object • `glassfish.embedded.tmpdir` system property value • `java.io.tmp` system property value • `as-install/domains/domain1` if a non-embedded installation is referenced
Creates a new configuration file or references an existing configuration file	`setConfigFileURI(String configFileURI)`	In order of precedence: • `domain-dir/config/domain.xml` if domain-dir was set using `setInstanceRoot` • built-in embedded `domain.xml`
Specifies whether the configuration file is read-only	`setConfigFileReadOnly(boolean readOnly)`	`true`

Purpose	Method	Default Value
Sets the port on which Embedded Payara Server listens.	`setPort`(String networkListener, int port)	None

NOTE Do not use `setPort` if you are using the `setInstanceRoot` or `setConfigFileURI` methods.

Creating an Embedded Payara Server

This example shows code for creating an Embedded Payara Server.

```
import org.glassfish.embeddable.*;

public class EmbeddedRunner{

    public static void main(String... args){
        var server = GlassFishRuntime.bootstrap().newGlassFish();
        server.start();
    }
}
```

Creating an Embedded Payara Server with configuration customizations

This example shows code for creating an Embedded Payara Server using the existing domain-dir `C:\samples\test\applicationserver\domains\domain1`:

```
import org.glassfish.embeddable.*;

public class EmbeddedRunner{
    public static void main(String... args){

        var bootstrapProperties = new BootstrapProperties();
        bootstrapProperties.setInstallRoot("C:\\samples\\test\\applicationserver");
        GlassFishRuntime glassfishRuntime = GlassFishRuntime.bootstrap
(bootstrapProperties);

        var glassfishProperties = new GlassFishProperties();
        glassfishProperties.setInstanceRoot
("C:\\samples\\test\\applicationserver\\domains\\domain1");
        var server = glassfishRuntime.newGlassFish(glassfishProperties);

        server.start();
    }
}
```

Running an Embedded Payara Server

After you create an embedded Payara Server as described in "Creating and Configuring an Embedded Payara Server", you can perform operations such as:

- "Setting the Port of an Embedded Payara Server From an Application"
- "Starting an Embedded Payara Server From an Application"
- "Stopping an Embedded Payara Server From an Application"

Setting the Port of an Embedded Payara Server From an Application

You must set the server's HTTP or HTTPS port. If you do not set the port, your application fails to start and throws an exception. You can set the port directly or indirectly.

> **NOTE** Do not use `setPort` if you are using `setInstanceRoot` or `setConfigFileURI`. These methods set the port indirectly.

- To set the port directly, invoke the `setPort` method of the `GlassFishProperties` object.
- To set the port indirectly, use a `domain.xml` file that sets the port. For more information, see "The `domain.xml` File".

This example shows code for setting the port of an embedded Payara Server.

```
import org.glassfish.embeddable.*;

public class EmbeddedRunner{
    public static void main(String... args){
        var glassfishProperties = new GlassFishProperties();
        glassfishProperties.setPort("http-listener", 8080);
        glassfishProperties.setPort("https-listener", 8181);
    }
}
```

Starting an Embedded Payara Server From an Application

To start an embedded Payara Server, invoke the `start` method of the corresponding

server object.

This example shows code for setting the port and starting an embedded Payara Server:

```
import org.glassfish.embeddable.*;

public class EmbeddedRunner{
    public static void main(String... args){
        var glassfishProperties = new GlassFishProperties();
        glassfishProperties.setPort("http-listener", 8080);
        glassfishProperties.setPort("https-listener", 8181);

        var server = GlassFishRuntime.bootstrap().newGlassFish(
glassfishProperties);
        server.start();
    }
}
```

Stopping an Embedded Payara Server From an Application

The API for an embedded Payara Server provides a method for stopping an embedded server. Using this method enables your application to stop the server in an orderly fashion by performing any necessary cleanup steps before stopping the server, for example:

- Un-deploying deployed applications
- Releasing any resources that your application uses

To stop an embedded Payara Server, invoke the `stop` method of an existing server object.

This example shows code for prompting the user to press the Enter key to stop an embedded Payara Server:

```
import java.io.BufferedReader;

import org.glassfish.embeddable.*;

public class EmbeddedRunner{
    public static void main(String... args){
        System.out.println("Press Enter to stop server");
        // wait for Enter
        server.stop();
    }
```

```
}
```

As an alternative, you can use the `dispose` method to stop an embedded Payara Server and dispose of the temporary file system.

Deploying and Un-Deploying an Application in an Embedded Payara Server

Deploying an application installs the files that comprise the application into Embedded Payara Server and makes the application ready to run. By default, an application is enabled when it is deployed.

To Deploy an Application from an Archive File or a Directory

An archive file contains the resources, deployment descriptor, and classes of an application. The content of the file must be organized in the directory structure that the Jakarta EE specifications define for the type of archive that the file contains. For more information, see deploying-applications in the Payara Server Application Deployment section.

Deploying an application from a directory enables you to deploy an application without the need to package the application in an archive file. The contents of the directory must match the contents of the expanded Jakarta EE archive file as laid out by the Payara Server. The directory must be accessible to the machine on which the deploying application runs. For more information about the requirements for deploying an application from a directory, see to-deploy-an-application-or-module-in-a-directory-format in the Payara Server Application Deployment section.

If some resources needed by an application are not under the application's directory, see "Creating a Scattered Archive".

1. Instantiate the `java.io.File` class to represent the archive file or directory.
2. Invoke the `getDeployer` method of the server object to get an instance of the `org.glassfish.embeddable.Deployer` class.
3. Invoke the `deploy`(`File`, `archive`,`params`) method of the instance of the `Deployer` object.

Specify the `java.io.File` class instance you created previously as the first method parameter.

For information about optional parameters you can set, see the descriptions of the "`deploy`" subcommand parameters. Simply quote each parameter in the method, for example `"--force=true"`.

This example shows code for deploying an application from the archive file `c:\samples\simple.war` and setting the name, context root, and force parameters.

```
import java.io.File;
import org.glassfish.embeddable.*;

public class EmbeddedRunner{
    public static void main(String... args){
        var glassfishProperties = new GlassFishProperties();
        glassfishProperties.setPort("http-listener", 8080);
        glassfishProperties.setPort("https-listener", 8181);

        var server = GlassFishRuntime.bootstrap().newGlassFish(
glassfishProperties);
        server.start();

        var war = new File("c:\\samples\\simple.war");
        var deployer = server.getDeployer();
        deployer.deploy(war, "--name=simple", "--contextroot=simple", "--
force=true");
        // deployer.deploy(war) can be invoked instead. Other parameters are
optional.
    }
}
```

Un-Deploying an Application

Undeploy an application when the application is no longer required to run in Payara Server. For example, before stopping Payara Server, undeploy all applications that are running in Payara Server.

To undeploy an application, invoke the `undeploy` method of an existing `Deployer` object. In the method invocation, pass the name of the application as a parameter. This name is specified when the application is deployed.

For information about optional parameters you can set, see the descriptions of the "`deploy`" command parameters. Simply quote each parameter in the method, for

example `"--cascade=true"`.

To undeploy all deployed applications, invoke the `undeployAll` method of an existing `EmbeddedDeployer` object. This method takes no parameters.

This example shows code for un-deploying the application that was deployed in the previous example:

```
import org.glassfish.embeddable.*;

public class EmbeddedRunner{
    public static void main(String... args){
        deployer.undeploy(war, "--droptables=true", "--cascade=true");
    }
}
```

Creating a Scattered Archive

Deploying a module from a scattered archive (WAR or JAR) enables you to deploy an un-packaged module whose resources, deployment descriptor, and classes are in any location. Deploying a module from a scattered archive simplifies the testing of a module during development, especially if all the items that the module requires are not available to be packaged.

In a scattered archive, these items are not required to be organized in a specific directory structure. Therefore, you must specify the location of the module's resources, deployment descriptor, and classes when deploying the module.

To create a scattered archive, perform these tasks:

1. Instantiate the `org.glassfish.embeddable.archive.ScatteredArchive` class.
2. Invoke the `addClassPath` and `addMetadata` methods if you require them.
3. Invoke the `toURI` method to deploy the scattered archive.

The methods of this class for setting the scattered archive configuration are listed in the following table. The default value of each configuration setting is also listed.

Table 179. Constructors and Methods of the `ScatteredArchive` Class

Purpose	Method	Default Value
Creates and names a scattered archive	`ScatteredArchive(String name, ScatteredArchive.Type type)`	None
Creates and names a scattered archive based on a top-level directory. If the entire module is organized under the topDir, this is the only method necessary. The `topDir` argument can be null if other methods specify the remaining parts of the module.	`ScatteredArchive(String name, ScatteredArchive.Type type, File topDir)`	None
Adds a directory to the classes classpath	`addClassPath(File path)`	None
Adds a metadata locator	`addMetaData(File path)`	None
Adds and names a metadata locator	`addMetaData(File path, String name)`	None
Gets the deployable URI for this scattered archive	`toURI()`	None

This example shows code for creating a WAR file and using the `addClassPath` and `addMetadata` methods. This example also includes the code from the previous example that showcases deploying an application from an archive file.

```
import java.io.File;
import org.glassfish.embeddable.*;

public class EmbeddedRunner{
    public static void main(String... args){
        var glassfishProperties = new GlassFishProperties();
        glassfishProperties.setPort("http-listener", 9090);

        var server = GlassFishRuntime.bootstrap().newGlassFish(
glassfishProperties);
        server.start();

        var deployer = glassfish.getDeployer();
        var archive = new ScatteredArchive("testapp", ScatteredArchive.Type.WAR);
        // target/classes directory contains complied servlets
        archive.addClassPath(new File("target", "classes"));
        // resources/payara-web.xml is the WEB-INF/payara-web.xml
        archive.addMetadata(new File("resources", "payara-web.xml"));
        // resources/web.xml is the WEB-INF/web.xml
```

```
        archive.addMetadata(new File("resources", "web.xml"));

        // Deploy the scattered web archive.
        var appName = deployer.deploy(archive.toURI(), "--contextroot=hello");

        deployer.undeploy(appName);
        server.stop();
        server.dispose();
    }
}
```

Creating a Scattered Enterprise Archive

Deploying an application from a scattered enterprise archive (EAR) enables you to deploy an un-packaged application whose resources, deployment descriptor, and classes are in any location. Deploying an application from a scattered archive simplifies the testing of an application during development, especially if all the items that the application requires are not available to be packaged.

In a scattered archive, these items are not required to be organized in a specific directory structure. Therefore, you must specify the location of the application's resources, deployment descriptor, and classes when deploying the application.

To create a scattered enterprise archive, perform these tasks:

1. Instantiate the `org.glassfish.embeddable.archive.ScatteredEnterpriseArchive` class.
2. Invoke the `addArchive` and `addMetadata` methods if you require them.
3. Invoke the `toURI` method to deploy the scattered enterprise archive.

The methods of this class for setting the scattered enterprise archive configuration are listed in the following table. The default value of each configuration setting is also listed.

Table 180. Constructors and Methods of the `ScatteredEnterpriseArchive` Class

Purpose	Method	Default Value
Creates and names a scattered enterprise archive	`ScatteredEnterpriseArchive(String name)`	None
Adds a module or library	`addArchive(File archive)`	None
Adds a module or library	`addArchive(File archive, String name)`	None
Adds a module or library	`addArchive(URI URI)`	None
Adds a module or library	`addArchive(URI URI, String name)`	None
Adds a metadata locator	`addMetaData(File path)`	None
Adds and names a metadata locator	`addMetaData(File path, String name)`	None
Gets the deployable URI for this scattered archive	`toURI()`	None

This example shows code for creating an EAR file and using the `addArchive` and `addMetadata` methods.

```
import java.io.File;
import org.glassfish.embeddable.*;

public class EmbeddedRunner{
    public static void main(String... args){
        var glassfishProperties = new GlassFishProperties();
        glassfishProperties.setPort("http-listener", 9090);

        var server = GlassFishRuntime.bootstrap().newGlassFish(
glassfishProperties);
        server.start();

        var deployer = glassfish.getDeployer();

        // Create a scattered web application.
        var webmodule = new ScatteredArchive("testweb", ScatteredArchive.Type.WAR);
        // target/classes directory contains my complied servlets
        webmodule.addClassPath(new File("target", "classes"));
        // resources/payara-web.xml is my WEB-INF/payara-web.xml
        webmodule.addMetadata(new File("resources", "payara-web.xml"));

        // Create a scattered enterprise archive.
        var archive = new ScatteredEnterpriseArchive("testapp");
```

```
        // src/application.xml is my META-INF/application.xml
        archive.addMetadata(new File("src", "application.xml"));
        // Add scattered web module to the scattered enterprise archive.
        // src/application.xml references Web module as "scattered.war".
        //Hence specify the name while adding the archive.
        archive.addArchive(webmodule.toURI(), "scattered.war");
        // lib/mylibrary.jar is a library JAR file.
        archive.addArchive(new File("lib", "mylibrary.jar"));
        // target/ejbclasses contain my compiled EJB module.
        // src/application.xml references EJB module as "ejb.jar".
        //Hence specify the name while adding the archive.
        archive.addArchive(new File("target", "ejbclasses"), "ejb.jar");

        // Deploy the scattered enterprise archive.
        var appName = deployer.deploy(archive.toURI());

        deployer.undeploy(appName);
        server.stop();
        server.dispose();
    }
}
```

Running `asadmin` Commands Using the Payara Server Embedded Server API

Running commands from an application enables the application to configure the embedded Payara Server to suit the application's requirements. For example, an application can run the required `asadmin` commands to create a JDBC technology connection to a database.

NOTE	Ensure that your application has started an embedded Payara Server before the application attempts to run any `asadmin` commands. For more information, see "Running an Embedded Payara Server".

The `org.glassfish.embeddable` package contains classes that you can use to run `asadmin` commands. Use the following code examples as templates and change the command name, parameter names, and parameter values as needed.

This example shows code for running an `asadmin create-jdbc-resource` command:

```
import org.glassfish.embeddable.*;

public class EmbeddedRunner{
    public static void main(String... args){
        var command = "create-jdbc-resource";
        var poolId = "--connectionpoolid=DerbyPool";
        var dbname = "jdbc/DerbyPool";
```

```
        var commandRunner = glassfish.getCommandRunner();
        var commandResult = commandRunner.run(command, poolId, dbname);
    }
}
```

This example shows code for running an `asadmin set-log-level` command:

```
import org.glassfish.embeddable.*;

public class EmbeddedRunner{
    public static void main(String... args){
        var command = "set-log-level";
        var level = "javax.enterprise.system.container.web=FINE";
        var commandRunner = glassfish.getCommandRunner();
        var commandResult = commandRunner.run(command, level);
    }
}
```

Sample Applications

This example shows code for the following:

- Using the existing file `c:\myapp\embeddedserver\domains\domain1\config\domain.xml` and preserving this file when the application is stopped.
- Deploying an application from the archive file `c:\samples\simple.war`.

```
import java.io.File;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import org.glassfish.embeddable.*;

public class EmbeddedRunner {
    public static void main(String[] args) {
        var configFile = new File
("c:\\myapp\\embeddedserver\\domains\\domain1\\config\\domain.xml");
        var war = new File("c:\\samples\\simple.war");
        try {
            var glassfishRuntime = GlassFishRuntime.bootstrap();
            var glassfishProperties = new GlassFishProperties();
            glassfishProperties.setConfigFileURI(configFile.toURI());
            glassfishProperties.setConfigFileReadOnly(false);

            var server = glassfishRuntime.newGlassFish(glassfishProperties);
            server.start();

            var deployer = server.getDeployer();
            deployer.deploy(war, "--force=true");
        }
        catch (Exception ex) {
            ex.printStackTrace();
```

```
        }

        System.out.println("Press Enter to stop the server");
        try(var reader = new BufferedReader(new InputStreamReader(System.in
)).readLine()) {
            server.dispose();
            server.shutdown();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Using the EJB Embeddable API with Embedded Payara Server

Payara Server Embedded is not related to the EJB Embeddable API, but you can use these APIs together in some form. The EJB Embeddable API supports all EJB Lite features with addition of the EJB timer service and testing of EJB modules packaged in a WAR file.

For EJB modules in a WAR file (or an exploded directory), if a web application has one EJB module, and there are no other EJB modules in the classpath, those entries (libraries) are ignored. If there are other EJB modules, a temporary EAR file is created. For EJB modules in a WAR file to be tested, the client code must use EJB modules with interfaces or without annotations. Those EJB modules are not part of the classpath and can't be loaded by the client class loader.

To Use the EJB Embeddable API with Embedded Payara Server

1. To specify Payara Server as the Container Provider, include the `payara-embedded-all.jar` dependency in the class path of your embeddable EJB application.

   See "Setting the Class Path" and section 17.2 of the Jakarta Enterprise Beans specification.

2. Configure any required resources.

   TIP The `jdbc/__default` H2 database is preconfigured in all distributions of Azul Payara.

   If your embeddable EJB application uses Java Persistence, you do not need to

specify a JDBC resource.

See "Default Jakarta Persistence Data Source for Embedded Payara Server".

3. Invoke one of the `createEJBContainer` methods.

IMPORTANT	Do not deploy your embeddable EJB application or any of its dependent Jakarta EE modules before invoking one of the `createEJBContainer` methods. These methods perform deployment in the background and do not load previously deployed applications or modules.

4. To change the "Instance Root Directory", set the `org.glassfish.ejb.embedded.glassfish.instance.root` system property value by using the `createEJBContainer(Map<?, ?> properties)` method.

   The default root directory location is `as-install/domains/domain1` if a non-embedded installation is referenced. This system property applies only to embeddable EJB applications used in Payara Server.

5. Close the EJB container properly to release all acquired resources and threads.

EJB Embeddable API Properties

Properties that can be passed to the `EJBContainer#createEJBContainer(Properties)` method are summarized in the following table.

NOTE	All properties use the `org.glassfish.ejb.embedded.glassfish` prefix. For example, the full name of the `installation.root` property is `org.glassfish.ejb.embedded.glassfish.installation.root`.

Table 181. EJB Embeddable API Properties

Property	Default	Description
`installation.root`	Payara Server installation location	The "Installation Root Directory".
`instance.root`	In order of precedence: • `glassfish.embedded.tmpdir` property value specified in `GlassFishProperties` object • `glassfish.embedded.tmpdir` system property value • `java.io.tmp` system property value • `as-install/domains/domain1` if a non-embedded installation is referenced	The "Instance Root Directory".
`configuration.file`	`domain-dir/config/domain.xml`	The server's configuration file.
`keep-temporary-files`	`false`	If `true`, keeps temporary files (exploded EAR file and configuration file) created by the embedded EJB container when Embedded Payara Server is stopped.
`web.http.port`	None	Enables the web container if set. Needed for testing web services in a WAR file. The value is ignored and can be an empty string.

Property	Default	Description
`instance.reuse`	`false`	If `true`, no changes are made to the existing configuration file, and a temporary server instance is not created for the embedded run. Instead, execution happens against the existing server instance. Do not use this option if the reused server instance could be in use by the running non-embedded Payara Server.
`skip-client-modules`	`false`	If `true`, omits modules from the classpath if they are not specified using `EJBContainer.MODULES` and have a manifest file with a `Main-Class` attribute.

Changing Log Levels in Payara Server Embedded

To change the JUL (Java Util Logging) log levels in Payara Server Embedded, you can follow the steps in this section, or you can use the Embedded Server API as shown in the previous examples.

For more information about Payara Server logging, see administering-the-logging-service in the Payara Server General Administration section.

You can change log levels in Payara Embedded Server in either of the following ways:

- Using the Payara Server Embedded Server API
- Creating a custom logging configuration file

Both these ways require logger names. For a list of logger names, use the "`list-log-levels`" subcommand.

This example shows how to set log levels using the `getLogger` method in the API:

```
import org.glassfish.embeddable.*;

public class EmbeddedRunner{

    public static void main(String[] args){
        // Create Embedded server
        var server = GlassFishRuntime.bootstrap().newGlassFish();

        // Set the log levels. For example, set 'deployment' log levels to FINEST
        Logger.getLogger("javax.enterprise.system.tools.deployment").setLevel(
Level.FINEST);

        // Start Embedded server and deploy an application.
        // You will see all the FINEST logs printed on the console.
        server.start();
        server.getDeployer().deploy(new File("sample.war"));

        // Dispose Embedded server
        server.dispose();
    }
}
```

This example shows the contents of a custom logging configuration file, `custom-logging.properties`.

```
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = FINEST
javax.enterprise.system.tools.deployment.level = FINEST
javax.enterprise.system.level = FINEST
```

Pass the name of this custom logging configuration file to the `java` command when you invoke the application starts the embedded server. For example:

```
java -Djava.util.logging.config.file=custom-logging.properties
fish.payara.embedded.ServerRunner
```

Default Jakarta Persistence Data Source for Embedded Payara Server

The `jdbc/__default` H2 database is preconfigured in any Embedded Payara Server. It is used when an application is deployed in Embedded Payara Server that uses Jakarta Persistence but doesn't specify a data source.

Embedded Payara Server uses an embedded H2 database created in a temporary domain that is destroyed when an embedded Payara Server is stopped. You can use an

H2 database configured with non-embedded Payara Server if you explicitly specify the instance root directory or through its configuration file.

Restrictions for Payara Server Embedded Web Profile

The web profile variant of the Embedded API (`payara-embedded-web.jar`) only supports the following Payara Server features:

- The following web technologies of the Jakarta EE platform:
  - Jakarta Servlet
  - Jakarta Server Pages (JSP) technology
  - Jakarta Faces technology
- JDBC-technology connection pooling
- Jakarta Persistence
- Jakarta Transactions

The full profile variant of the Embedded API (`payara-embedded-all.jar`) support all features of Payara Server with the following exceptions:

- Administration Console
- Upgrade Tool (Enterprise-edition only)
- Apache Felix OSGi framework

Since an Embedded Payara Server requires no installation or configuration, the following files and directories are absent from the file system until the server is properly started:

- `default-web.xml` file
- `domain.xml` file
- Applications directory
- Instance root directory

When embedded Payara Server is started, the base installation directory that Payara

Server uses depends on the options with which Payara Server is started. If necessary, an embedded Payara Server creates a base installation directory and then copies the following directories and their contents from the Java archive (JAR) file in which embedded Payara Server is distributed:

- `domains`
- `lib`

NOTE | If necessary, Payara Server also creates an instance root directory.

Payara Server Embedded Use Cases

Overview of some expected use cases of Payara Server Embedded.

Bundle Your Application

Where the installation of Payara Server and the deployment of the applications can't be done on a central infrastructure, you may opt to use Payara Embedded. In this case, you can bundle Payara Server with the application within a single application, which makes it easier for installing it.

Instead of having the three steps, installing Payara Server, configuring the environment, and deploying the application, you can do all those steps within the application containing Payara Embedded. You only need to start the application and the system is ready to respond to user requests.

Integration Testing

While performing integration testing, you may need to deploy the application to Payara in order to verify the behaviour of the code you have written. While you could use a standalone installation of Payara Server, cleanup is not always done correctly and can cause other tests to behave incorrectly.

Integration testing can be done easier and more consistently with Payara Embedded, you can start Payara up as the beginning of the test, verify the functionality and then use a clean instance for each test, therefore eliminating the possibility of an unexpected state in the environment.

Starting Payara Embedded for testing can also be done using the Arquillian connector. In that case, the setup and teardown of Payara is managed for you.

JDK 17 Considerations

When running Payara Embedded on JDK 17, you must update your client side run configuration with all the necessary opens and exports for Payara to work as expected. The following JVM options should be added within your run configuration:

```
--add-opens=java.base/jdk.internal.loader=ALL-UNNAMED --add-opens
=jdk.management/com.sun.management.internal=ALL-UNNAMED --add-exports
=java.base/jdk.internal.ref=ALL-UNNAMED --add-opens=java.base/java.lang=ALL-UNNAMED
--add-opens=java.base/java.nio=ALL-UNNAMED --add-opens=java.base/sun.nio.ch=ALL-
UNNAMED --add-opens=java.management/sun.management=ALL-UNNAMED --add-opens
=java.base/sun.net.www.protocol.jrt=ALL-UNNAMED -Xbootclasspath/a:${
com.sun.aas.installRoot}/lib/grizzly-npn-api.jar --add-exports
=java.base/sun.net.www=ALL-UNNAMED --add-exports=java.base/sun.security.util=ALL-
UNNAMED --add-opens=java.base/java.lang.invoke=ALL-UNNAMED --add-opens
=java.desktop/java.beans=ALL-UNNAMED --add-exports=jdk.naming.dns/com.sun.jndi.dns
=ALL-UNNAMED --add-opens=java.base/sun.net.www.protocol.jar=ALL-UNNAMED
```

See Also

- "Payara Arquillian Connector"

Application Development

Overview

This chapter describes how to create and run Jakarta EE applications in compatible Azul Payara distributions. Topics for this guide include developer tools, security, debugging and specific features for Jakarta EE specification implementations. This document is intended for use by software developers who create, assemble, and deploy Jakarta EE applications using Azul Payara.

Related Documentation

The following tutorials explain how to develop Jakarta EE applications:

- Your First Cup: An Introduction to the Jakarta EE Platform. This is a recommended tutorial for programmers with no experience or knowledge about Jakarta EE, and it explains the entire process for developing a simple enterprise application using

Eclipse Glassfish.

- The Jakarta EE Tutorial. This comprehensive tutorial explains how to use Jakarta EE platform technologies and APIs to develop enterprise applications. This guide heavily references the contents of this tutorial to guide users in understanding the basic concepts of application development.

IMPORTANT	The Jakarta EE tutorial uses Eclipse Glassfish as its Jakarta EE runtime. Most of the concepts and instructions used in the tutorial should be directly equivalent to Azul Payara given its similarities to Eclipse Glassfish.

The Jakarta EE specifications Javadoc documentation is listed within its specification resources under each corresponding specification site at https://jakarta.ee/specifications/.

The Jakarta EE Examples project is a collection of sample applications that demonstrate a broad range of Jakarta EE technologies. They are available to download in their own online repository.

Class Loaders

NOTE	This section strictly applies to Payara Server.

Payara Server extends the default JVM classloading and provides additional classes, which are loaded by the class loaders included by default on this distribution. Understanding Payara Server class loaders can help you determine where to place supporting JAR and resource files for your modules and applications.

Understanding Payara Server class loaders can help you determine where to place supporting JAR and resource files for your modules and applications.

In a JVM implementation, the class loaders dynamically load a specific Java class file needed for resolving a dependency. For example, when an instance of `java.util.Enumeration` needs to be created, one of the class loaders loads the

relevant class into the environment.

For information about class loader debugging, see class-loader-debugging.

Standard Classloading

In a JVM implementation, the class loaders dynamically load a specific Java class file needed for resolving a dependency. For example, when an instance of `java.util.Enumeration` needs to be created, one of the class loaders loads the relevant class into the environment.

For information about class loader debugging, see class-loader-debugging.

Class loaders in Payara Server follow a delegation hierarchy by default.

The Class Loader Hierarchy

Class loaders in the Payara Server runtime follow a delegation hierarchy that is illustrated in the following table:

Class Loader	Description
Bootstrap	The Bootstrap class loader loads the basic runtime classes provided by the JVM.
Extension	The Extension class loader loads classes from JAR files present in the system extensions directory, `domain-dir/lib/ext` (that is, libraries of type `ext`). It's parent to the Public API class loader. **WARNING** The Extension classloader no longer works using the Java Optional Package mechanism (see http://docs.oracle.com/javase/8/docs/technotes/guides/extensions/extensions.html) since this mechanism is no longer supported in Java 11+. Instead, this classloader is designed to "trick" the server in handling delegation much sooner.

Class Loader	Description
Public API	The Public API class loader makes available all classes specifically exported by the Payara Server runtime for use by deployed applications. This includes, but is not limited to, Jakarta EE API, Eclipse MicroProfile and the Azul Payara Public API. Parents the Common class loader.
Common	The Common class loader loads JAR files in the`as-install/lib` directory, followed by JAR files in the `domain-dir/lib` directory. Using `domain-dir/lib` is recommended whenever possible, and required for custom login modules and security realms. Parents the Connector class loader.
Connector	The Connector class loader is a single class loader instance that loads individually deployed connector modules, which are shared across all applications. Parents the **Applib** class loader.
Applib	The `Applib` class loader loads the library classes (specifically, libraries of the `app` type), specified during deployment, for a specific enabled module or Jakarta EE application; One instance of this class loader is present in each class loader universe; Parents the Archive class loader. **NOTE** | When multiple deployed applications use the same library, they share the same instance of the library. One library cannot reference classes from another library.
Archive	The Archive class loader loads classes from the WAR, EAR, and JAR files or directories (for directory deployment) of applications or modules deployed in Payara Server. This class loader also loads any application-specific classes generated by the Payara Server runtime, such as stub classes or servlets generated by JSP pages.

Classloaders that are lower in the hierarchy will first delegate searching for a class to a classloader higher in the hierarchy.

NOTE	If none of the classloaders higher in the hierarchy finds the class, the classloader will attempt to find the class itself.

Here, you can see that:

- Classes provided by Payara Server have higher priority than classes provided by an application
- Classes provided by Payara Server have higher priority than installed libraries of type `common` and `app`.
- Classes in installed libraries have higher priority than classes provided by an application
- To install a library with a higher priority than classes provided by Payara Server, it should be installed with the `ext` type.

Configuring Delegation

The Jakarta Servlet specification recommends that a web module's class loader look in the local class loader before delegating to its parent.

You can make this class loader follow the delegation inversion model in the Servlet specification by setting `delegate="false"` in the `class-loader` element of the `payara-web.xml` file. It is safe to do this only for a web module that does not interact with any other modules.

For more details, see "class-loader" in the Payara Server Application Deployment section. An example of how to use this element is given in the disabling classloading delegation on web application section.

The default value is `delegate="true"`, which causes a web module's class loader to delegate in the same manner as the other class loaders. You must use `delegate="true"` for a web application that accesses EJB components or that acts

as a web service client or endpoint.

For a number of packages, including `java.*` and `jakarta.\*`, symbol resolution is always delegated to the parent class loader regardless of the `delegate` setting. This prevents applications from overriding core Java runtime classes or changing the API versions of specifications that are part of the Jakarta EE platform.

The delegation hierarchy can also be tweaked using the enhanced classloading features. For example, it's possible to give the classes provided by an application the highest priority (by disabling classloading delegation), or completely hide (isolate) classes provided by Payara Server from a deployed application.

Class Loader Universes

Access to components within applications and modules installed on the server occurs within the context of isolated class loader universes, each of which has its own *Applib* and *Archive* classloaders.

Application Universe: Each application has its own class loader universe, which loads the classes in all the modules in the application.

Individually Deployed Module Universe: Each individually deployed EJB JAR or web WAR has its own class loader universe, which loads the classes in the module.

A resource such as a file that is accessed by a Jakarta EE application component must be in one of the following locations:

- A directory pointed to by the Libraries field or `--libraries` option used during deployment
- A directory pointed to by the `library-directory` element in the `application.xml` deployment descriptor
- A directory pointed to by the application or module's classpath; for example, a web module's classpath includes these directories:

```
module-name/WEB-INF/classes
```

```
module-name/WEB-INF/lib
```

Installing Libraries in a Domain

Since each application or individually deployed module class loader universe is isolated, an application or module cannot load classes from another application or module. *This prevents two similarly named classes in different applications or modules from interfering with each other.*

To circumvent this limitation for libraries, utility classes, or individually deployed modules accessed by more than one application, you can include the relevant path to the required classes by installing a library with those classes into the server domain. This is useful if you'd like to build an application for flexible deployments and configure it with classes dropped on the classpath, or if you'd like multiple deployed applications to share the same library.

If an internal server resource, such as a JDBC connection pool, requires an additional library, you can install it directly into the Payara Server domain too.

To install a library into a server domain, use the asadmin add-library command.

This command accepts the `--type` argument, which accepts the following options:

`common`: The library is installed as a common library into the `lib` directory in the Payara Server domain **`ext`**: The library is installed as an "extension" library into the `lib/ext` directory in the Payara Server domain **`app`**: the library is installed as an application library into the `lib/applibs` directory in the Payara Server domain **ifeval**: ["enterprise" == "community"] **`war`**: the library is installed as a WAR library into the `lib/warlibs` directory in the Payara Server domain **endif**: []

Common Libraries

Common libraries are available to all applications or modules deployed on servers with the same configuration. There are several levels of common libraries:

- Shared by all applications deployed on the same Payara Server installation - located in the `glassfish/lib` directory in the Payara Server installation

- Shared by all applications deployed in the same Payara Server domain - located in the `lib` directory in that Payara Server domain directory
- Shared by all applications deployed on instances that share the same configuration - located in the `config/<config-name>/lib` directory in the particular Payara Server domain directory

NOTE	Only libraries shared by applications in the same domain can be installed by the `add-library` asadmin command. All other types have to be installed manually by copying the libraries into the particular locations.

Extension Libraries

Application developers can use libraries installed as extension libraries to extend the functionality of the core Payara Server platform. For example, an Oracle Database JDBC driver should be installed as an extension library if it's going to be used via JPA to use Oracle-specific JPA features.

Or a Java agent library should be installed as an extension library if the Java agent instruments classes in Payara Server to use classes from the agent library itself.

CAUTION	Classes in extension libraries will be available to all deployed applications. If you need that classes from an extension library are available also to Payara Server's internal classes (and OSGi bundles in general) as is often the case with Java agents, you also need to add the packages exported by the library into OSGi boot delegation.

This is done by modifying the `as-install/config/osgi.properties` configuration file in the Payara Server installation and adding the packages to the list in the `org.osgi.framework.bootdelegation` property.

TIP	Packages exported by Oracle JDBC drivers are already added by default.

Extension libraries are added to the system classpath at the beginning of the classpath, so that they have higher priority than all other classes and libraries on the classpath.

This is because the Java extension mechanism isn't supported in Java 11 and newer and the official recommend way to replace it is to add the libraries to the classpath directly.

Since defining the `java.ext.dirs` on Java 11+ leads to an error, this JVM option cannot be used with Java 11+ and extension libraries are automatically added as classpath elements.

Application Specific Libraries

You can specify module or application-specific library classes in one of the following ways:

- Use the Administration Console. Open the *Applications* component, then go to the page for the type of application or module. Select the *Deploy* button. Type the comma-separated paths in the *Libraries* field.
- Use the `asadmin deploy` command with the `--libraries` option and specify comma-separated paths. For more details, see the "Payara Server Reference Manual".
- Use the `asadmin add-library` command with the `--type app` option. For details, see the "add-library".

NOTE	None of these alternatives apply to application clients. For more information, see using-libraries-with-application-clients.

You can update a library JAR file using dynamic reloading or by restarting (disabling and re-enabling) a module or application. To add or remove library JAR files, you can redeploy the module or application. Application libraries are included in the *Applib* class loader. Paths to libraries can be relative or absolute.

A relative path is relative to `domain-dir/lib/applibs`. If the path is absolute, the path must be accessible to the domain administration server (DAS). Payara Server automatically synchronizes these libraries to all remote instances when a cluster or deployment group is restarted. However, libraries specified by absolute paths are not

guaranteed to be synchronized.

TIP	You can also use application-specific class loading to access different versions of a library from different applications.

If multiple applications or modules refer to the same libraries, classes in those libraries are automatically shared. This can reduce the memory footprint and allow sharing of static information. However, applications or modules using application-specific libraries are not portable.

NOTE	If you see an access control error message when you try to use a library, you may need to grant permission to the library in the `server.policy` file. For more information, see changing-permissions-for-an-application.

ifeval: ["enterprise" == "community"]

WAR Libraries

Libraries stored in this directory allows a developer to create and deploy skinny WARs, because they are subject to additional annotation and extension scanning as well as processing for CDI, EJB, Servlet, and more. Unlike common and application libraries (stored in `domain_dir/lib` and `domain_dir/lib/applibs` respectively), warlibs are cached and don't dynamically update if the files within the directory are modified.

These libraries are only used by applications that are deployed with the `warlibs` property set to `true`. See the "deploy" command documentation for additional information.

endif: []

Packaging the Client JAR for One Application in Another Application

By packaging the client JAR for one application in a second application, you allow a Jakarta EE component in the second application to call an EJB component in the first (dependent) application, without making either of them accessible to any other application or module.

TIP	As an alternative for a production environment, you can have the Common class loader load the client JAR of the dependent application as described in the previous section.

To do this follow these steps:

1. Deploy the dependent application.
2. Add the dependent application's client JAR file to the calling application.
   - For a calling EJB component, add the client JAR file at the same level as the EJB component. Then add a `Class-Path` entry to the `MANIFEST.MF` file of the calling EJB component.
   - For a calling web component, add the client JAR file under the `WEB-INF/lib` directory.

     If you need to package the client JAR with both the EJB and web components, set `delegate="true"` attribute in the `class-loader` element of the `payara-web.xml` file.

     This changes the Web class loader so that it follows the standard class loader delegation model and delegates to its parent before attempting to load a class itself.

NOTE	For most applications, packaging the client JAR file with the calling EJB component is sufficient. You do not need to package the client JAR file with both the EJB and web components unless the web component is directly calling the EJB component in the dependent application.

3. Deploy the "client" application. The calling EJB or web component must specify in its `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` or `payara-web.xml` file the JNDI name of the EJB component in the dependent application. Using an `ejb-link` mapping does not work when the EJB component being called resides in another

application.

Enhanced Classloading

This sections below cover the enhanced class loading functionality provided by Azul Payara.

Default Class and Library Loading

Payara Server has included many standard Java libraries and packages, for example **Jackson**, **Nimbus JOSE**, **Logback**, and others to use. These libraries are located on the `as-install/modules` directory.

The default class loading mechanism of Payara Server works like this: When loading classes that belong to a library or framework that is included in the server, the server **will always** load those classes even if the application itself includes different versions.

In some cases, application developers will want to include a different version of the libraries that are already included on the server. Common use cases for this are:

1. Use a newer version of a library that is included in the server. For example, Payara Server includes the Jackson library, and you might need to use a newer version that includes a specific feature you want to use.
2. Use an older version of a library included within the server in order to support legacy applications. For example, you are using an older version of **Icefaces** that depends on an older version of Jakarta Faces.

Unfortunately, due to the way the default class loading hierarchy works, this will not be possible, and all libraries included with the server libraries will take precedence.

Disable Classloading Delegation

As detailed in Configuring Delegation, it is possible to invert/disable the way that delegation works as per the requirements of the Jakarta Servlet specification. Azul Payara introducers greater delegation disabling mechanisms, detailed in the following sections.

Disable Classloading Delegation Globally

To disable class loading delegation globally, you can set the system property `fish.payara.classloading.delegate` to `false`.

Disable Classloading Delegation Locally

It's possible to disable class loading delegation directly at the application level. This can be done for both WAR and EAR applications.

For **Web** applications, you can include `<class-loader delegate="false"/>` element in the `payara-web.xml`/`glassfish-web.xml` deployment descriptor.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

Here's an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-web-app PUBLIC "-//Payara.fish//DTD Payara Server 7 Servlet 6.1
Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-web-app_6_1-1.dtd">
<payara-web-app error-url="">
  <class-loader delegate="false"/>
</payara-web-app>
```

For **EAR** applications, you can include the `<classloading-delegate>false</classloading-delegate>` element in the `payara-application.xml` deployment descriptor.

Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-application PUBLIC "-//Payara.fish//DTD Payara Application Server
7 Jakarta EE Application 11 Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-application_11-1.dtd">
<payara-application>
    <classloading-delegate>false</classloading-delegate>
</payara-application>
```

Extreme Classloading Isolation

It's possible to configure an extreme isolation level on the class loading delegation for deployed applications. With this extreme isolation behavior, a deployed application can force the server to load only classes and resources from libraries included on the server that belong to **whitelisted packages** defined on its deployment descriptors.

To configure whitelist packaging you can use the `<whitelist-package>` element on the `payara-web.xml` / `glassfish-web.xml` (for WAR artifacts) or the `glassfish-application.xml`/`payara-application.xml` (for EAR artifacts) deployment descriptors. This element can be included multiple times to whitelist multiple packages.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

Here is an example of whitelisting both the **Google Guava**, **Jackson** and **Faces Config** packages for a WAR application:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-web-app PUBLIC "-//Payara.fish//DTD Payara Server 7 Servlet 6.1
Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-web-app_6_1-1.dtd">
<payara-web-app error-url="">
  <whitelist-package>com.google.guava</whitelist-package>
  <whitelist-package>com.fasterxml.jackson</whitelist-package>
  <whitelist-package>com.sun.faces.spi.FacesConfigResourceProvider</whitelist-
package>
</payara-web-app>
```

The whitelist syntax is simple: Define the name of the package which contains the classes or resources in question. For example writing `com.google` would whitelist all Google libraries included on the server, while writing `com.google.guava` would only whitelist the Google Guava library instead.

IMPORTANT	To enable this extreme isolation behavior, at least one `whitelist-package` element must be defined in the appropriate descriptor.

Default Whitelisted Classes

Certain classes are whitelisted automatically, meaning they will always be loaded from the server libraries and follow the default classloading hierarchy, even if this feature is turned on.

This is because these packages are required by the server and therefore cannot be loaded from application modules:

- `java`
- `javax`
- `com.sun`
- `org.glassfish`
- `org.apache.jasper`
- `fish.payara`
- `com.ibm.jbatch`
- `org.hibernate.validator`
- `org.jboss.weld`
- `com.ctc.wstx`

Likewise, the default whitelisted resources are:

- `META-INF/services/jakarta.`
- `META-INF/services/org.glassfish.`
- `META-INF/services/java.`

Debugging Applications

This chapter gives guidelines for debugging applications deployed in Payara Server or Payara Micro.

Debugging in Payara Server

Enabling Debugging

When you enable debugging, you enable both local and remote debugging. To start the server in debug mode, use the `--debug` option as follows:

```
asadmin start-domain --debug [domain-name]
```

You can then attach to the server from the Java Debugger (`jdb`) at its default Java Platform Debugger Architecture (JPDA) port, which is `9009`.

For example, in UNIX systems:

```
jdb -attach 9009
```

And for Windows OS:

```
jdb -connect com.sun.jdi.SocketAttach:port=9009
```

For more information about the `jdb` debugger, see the following official Java SE 11 documentation links:

- Java Platform Debugger Architecture: `https://docs.oracle.com/en/java/javase/11/docs/specs/jpda/jpda.html`
- Java SE `jdb` command: `https://docs.oracle.com/en/java/javase/11/tools/jdb.html`

Payara Server debugging is based on JPDA. For more information, see JPDA Options.

You can attach any JPDA compliant debugger to Payara Server, including that of all major Java IDE environments like IntelliJ, Apache Netbeans, Eclipse and Visual Studio Code.

You can enable debugging even when Payara Server is started without the `--debug` option. This is useful in development environments where you want to make sure that

debugging is always turned on.

To Set the Server to Automatically Start up in Debug Mode

1. Use the Administration Console. Select the *JVM Settings* component under the relevant configuration.
2. Check the *Debug Enabled* box.
3. To specify a different port (from `9009`, the default) to use when attaching the JVM software to a debugger, specify `address=` port-number in the *Debug Options* field.
4. To add JPDA options, add any desired JPDA debugging options in Debug Options. See jpda-options.

JPDA Options

The default JPDA options in Payara Server are as follows:

```
-Xdebug -agentlib:transport=dt_socket,server=y,suspend=n,address=9009
```

> **TIP** In Windows OS, you can change `dt_socket` to `dt_shmem` to connect via a shared memory transport protocol.

If you specify `suspend=y`, the JVM software starts in suspended mode and stays suspended until a debugger attaches to it. This is helpful if you want to start debugging as soon as the JVM process starts.

To specify a different port (from `9009`, the default) to use when attaching the JVM software to a debugger, specify `address=<port-number>`.

You can include any additional options if needed.

Generating a Stack Trace for Debugging

To generate a Java stack trace for debugging purposes, use the `asadmin generate-jvm-report --type=thread` command. The generated stack trace goes to the `domain-dir/logs/server.log` file and also appears on the command prompt screen.

For more information about the `asadmin generate-jvm-report` command, see "`generate-jvm-report`".

Application Client Debugging

When the `appclient` script executes the `java` command to run the Application Client Container (ACC), which in turn runs the client itself, it includes on the command line the value of the `VMARGS` environment variable. You can set this variable to any suitable value. For example:

```
VMARGS=-agentlib:transport=dt_socket,server=y,suspend=y,address=8118
```

IMPORTANT	For debugging an application client, you must set the suspend option to `y` so you can connect the debugger to the client before any code has actually executed. Otherwise, the client may start running and execute past the point you want to examine.

You should use different ports for the server and client if you are debugging both concurrently. For details about setting the port, see JPDA Options.

You can also include JVM options in the `appclient` script directly. For information about the `appclient` script, see "`appclient`".

NOTE	The Application Client Container is supported only in the Full Profile, not in the Web Profile. See developing-java-clients.

Open Message Queue Debugging

Open Message Queue has a broker logger, which can be useful for debugging Java Message Service (JMS) applications, including message-driven bean applications.

You can adjust the logger's verbosity, and you can send the logger output to the broker's console using the broker's `-tty` option.

NOTE	JMS resources are supported only in the Full Profile, not in the Web Profile. See using-jakarta-messaging.

Enabling Verbose Mode

To have the server logs and messages printed to `System.out` on your command prompt screen, you can start the server in **verbose mode**. This makes it easy to do simple debugging using print statements, without having to check the `server.log` file every time.

To start the domain in verbose mode, use the `--verbose` option as follows:

```
asadmin start-domain --verbose [domain-name]
```

When the server is in verbose mode, messages are logged to the console or terminal window in addition to the log file. In addition, pressing `Ctrl-C` stops the server and pressing `Ctrl-\` (on UNIX platforms) or `Ctrl-Break` (on Windows OS) prints a thread dump.

On UNIX platforms, you can also print a thread dump using the `jstack` command or the command `kill -QUIT <process_id>`.

Class Loader Debugging

To generate class loading messages you can use the `-Xlog` JVM argument by running the following `asadmin create-jvm-options` command:

```
asadmin create-jvm-options -Xlog\:class+load
```

As classloading debugging messages will heavily populate the standard output, it is recommended to log these entries on a special log file. The `-Xlog` argument can be appended with the `file:[file-path]` option to store the information on a separate log file like this:

```
asadmin create-jvm-options -Xlog\:class+load\:file=classloading.log
```

It is recommended that this log is located in the standard logs directory output like so:

```
asadmin create-jvm-options -Xlog\:class+load\:file=${com.sun.aas.instanceRoot
}/logs/classloading.log
```

TIP

More information about how to configure the log levels of events to filter and other useful tags available to the `-Xlog` JVM argument cane be read on its command help output by running `java -Xlog:help`

Debugging in Payara Micro

As Payara Micro server instance is a standalone Java process that runs without an administration interface, debugging applications that run in it is as simple as configuring the debugging options when launching a new server instance from the command line:

```
java -agentlib:jdwp=transport=dt_socket,server=y,address=9011,suspend=y -jar
payara-micro.jar --deploy myapp.war
```

Then you can attach a new debugger session from within your preferred IDE.

Profiling Tools

You can use a profiler to perform remote profiling on Payara Server to discover bottlenecks in server-side performance. Profilers are extremely useful (either **Instrumental** or **Sampling** profiles) in analysing the CPU cycles of an application's specific components, finding bottlenecks on long-spanning operations, track the runtime of specific methods, etc.

Most profiling tools in the market can be integrated with Payara Server running on Java 11. The following "free-to_use" tools are recommended to be used due to their simplicity and ease to integrate with Payara Server:

Apache NetBeans Profiler: Comes bundled with Apache Netbeans, which has native support for running, debugging and profiling applications in Payara Server. For more information read Introduction to Profiling Java Applications in NetBeans IDE

Async Profiler: A powerful lightweight profiler that can run locally or remotely and trace CPU cycles, hardware and software performance and do Java Heap allocation analysis. This profiler is currently maintained as an open source project.

+ NOTE: If using IntelliJ IDEA Ultimate, Async profiler is bundled by default and requires no extra installation.

Securing Applications

This chapter describes how to write secure Jakarta EE applications, which contain components that perform user authentication and access authorization for the business logic of Jakarta EE components.

For general information about Jakarta EE security, see Security in The Jakarta EE Tutorial.

NOTE

Payara Server's Web Profile supports the EJB 4.0+ specification, which allows enterprise beans within web applications, among other features. The Full Profile supports the entire EJB 4.0+ specification. For details, see Jakarta Enterprise Beans.

Security Goals

In an enterprise computing environment, there are many security risks. The goal of Payara Server is to provide highly secure, interoperable, and distributed component computing based on the Jakarta EE security model. Security goals include:

- Full compliance with the Jakarta EE security model. This includes EJB and servlet role-based authorization.
- Full compliance with Eclipse MicroProfile JWT Propagation API.
- Support for single sign-on across all Payara Server applications within a single security domain.
- Support for web services message security.
- Security support for application clients.
- Support for the Jakarta Authentication API.
- Support for the Jakarta Authorization API.

- Support for Web Services Interoperability Technologies (WSIT) as described in the Metro Users Guide.
- Support for the Jakarta Security API
- Support for several underlying security realms, such as simple file, JDBC and Lightweight Directory Access Protocol (LDAP) configurable realms.

  Certificate authentication is also supported for Secure Socket Layer (SSL) client authentication.

NOTE	For Solaris, OS platform authentication is supported in addition to these.

- Support for declarative security through Payara Server specific XML-based role mapping.
- Support for OpenID Connect compatible security features (such as OAuth2).

Payara Server Specific Security Features

Payara Server supports the Jakarta EE Platform's security model (and added features courtesy of Eclipse MicroProfile APIs), as well as the following features which are specific to it:

- Message security; see Configuring Message Security for Web Services
- Single sign-on across all Payara Server applications within a single security domain; see User Authentication for Single Sign-on
- Programmatic login; see Programmatic Login Using the ProgrammaticLogin Class
- OAuth2 integration support.

Container Security

The component containers are responsible for providing Jakarta EE application security. The container provides two security forms:

- Declarative Security

- Programmatic Security

Annotations (also called metadata) enable a declarative style of programming, and so encompass both the declarative and programmatic security concepts. Users can specify information about security within a class file using annotations.

When an application is deployed, this information can either be used by or overridden by the application or module deployment descriptors.

Declarative Security

Declarative security means that the security mechanism for an application is declared and handled externally to the application.

Deployment descriptors describe the Jakarta EE application's security structure, including security roles, access control, and authentication requirements.

Payara Server supports the deployment descriptors specified by Jakarta EE and has additional security elements included in its own deployment descriptors. Declarative security is the application deployer's responsibility.

There are two levels of declarative security, as follows:

- Application Level Security
- Component Level Security

Application Level Security

For an application, roles used by any application must be defined in `@DeclareRoles` annotations in the code or `role-name` elements in the application deployment descriptor (`application.xml`).

These role names are scoped to the EJB XML deployment descriptors (`ejb-jar.xml` and `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` files) and to the servlet XML deployment descriptors (`web.xml` and `payara-web.xml` files).

For an individually deployed web or EJB module, you define roles using

`@DeclareRoles` annotations or `role-name` elements in the Jakarta EE deployment descriptor files `web.xml` or `ejb-jar.xml`.

To map roles to principals and groups, define matching `security-role-mapping` elements in the `glassfish-application.xml`/`payara-application.xml`, `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml`, or `glassfish-web.xml` /`payara-web.xml` file for each `role-name` used by the application.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

By default, group principal names are mapped to roles of the same name. Accordingly, the **Default Principal To Role Mapping** setting is enabled by default on the *Security* page of the Payara Server Administration Console.

TIP	This default role mapping definition is in effect if you do not define your own mapping in the deployment descriptor for your application as described in this section.

Component Level Security

Component level security encompasses web components and EJB components.

A secure web container authenticates users and authorizes access to a servlet or JSP by using the security policy laid out in the servlet XML deployment descriptors (`web.xml`, `glassfish-web.xml` and `payara-web.xml` files).

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

The EJB container is responsible for authorizing access to a bean method by using the security policy laid out in the EJB XML deployment descriptors (`ejb-jar.xml` and `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` files).

Programmatic Security

Programmatic security involves an EJB component or servlet using method calls to the security API, as specified by the Jakarta EE security model, to make business logic decisions based on the caller or remote user's security role.

Programmatic security should only be used when declarative security alone is insufficient to meet the application's security model.

The API for programmatic security consists of methods of the Jakarta Security API `SecurityContext` interface, and methods of the EJB `EJBContext` interface and the servlet `HttpServletRequest` interface.

Payara Server supports these interfaces as specified in the Jakarta EE specification.

There is also a proprietary Payara API for programmatic login. See Programmatic Login Using the ProgrammaticLogin Class.

For more information about programmatic security in web applications, see Using Programmatic Security with Web Applications in the Jakarta EE Tutorial.

Roles, Principals, and Principal to Role Mapping

By default, any groups that an authenticated user belongs to will be mapped to roles with the same names.

Therefore, the **Default Principal To Role Mapping** setting is enabled by default on the *Security* page of the Payara Administration Console. To change the default mapping you can clear this setting.

For applications, you define roles in `@DeclareRoles` annotations or the Jakarta EE deployment descriptor file `application.xml`. You define the corresponding role mappings in the Payara Server deployment descriptor file `glassfish-application.xml`/`payara-application.xml`.

For individually deployed web or EJB modules, you define roles in `@DeclareRoles` annotations or the Jakarta EE deployment descriptor files `web.xml` or `ejb-jar.xml`.

You define the corresponding role mappings in the Payara Server deployment descriptor files `glassfish-web.xml`/`payara-web.xml` or `glassfish-ejb-jar.xml` /`payara-ejb-jar.xml`.

For more information regarding Jakarta EE deployment descriptors, see the Jakarta EE Specification. For more information regarding Payara Server deployment descriptors, see "elements-of-the-payara-platform-deployment-descriptors" in the Application Deployment section.

Each `security-role-mapping` element in the `glassfish-application.xml` /`payara-application.xml`, `glassfish-web.xml`/`payara-web.xml`, or `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` file maps a role name permitted by the application or module to principals and groups.

For example, a `payara-web.xml` file for an individually deployed web module might contain the following:

```
<payara-web-app>
    <security-role-mapping>
        <role-name>manager</role-name>
        <principal-name>jgarcia</principal-name>
        <principal-name>mwebster</principal-name>
        <group-name>team-leads</group-name>
    </security-role-mapping>
    <security-role-mapping>
        <role-name>administrator</role-name>
        <principal-name>dsmith</principal-name>
    </security-role-mapping>
</payara-web-app>
```

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

A role can be mapped to either specific principals or to groups (or both). The principal or group names used must be valid principals or groups in the realm for the application or module.

Note that the `role-name` in this example must match the `@DeclareRoles` annotations or the `role-name` in the `security-role` element of the corresponding

`web.xml` file.

You can also specify a custom principal implementation class. This provides more flexibility in how principals can be assigned to roles.

A user's Jakarta Authentication login module now can authenticate its custom principal, and the authenticated custom principal can further participate in the Payara Server's authorization process. For example:

```
<security-role-mapping>
    <role-name>administrator</role-name>
    <principal-name class-name="CustomPrincipalImplClass">dsmith</principal-name>
</security-role-mapping>
```

You can specify a default principal and a default principal to role mapping, each of which applies to the entire Payara Server instance. The default principal to role-mapping maps group principals to the same named roles.

Web modules that omit the `run-as` element in `web.xml` use the default principal. Applications and modules that omit the `security-role-mapping` element use the default principal to role mapping. These defaults settings are part of the *Security Service*, which you can access in the following ways:

- In the Administration Console, select the *Security* component under the relevant configuration.
- Use the `asadmin set` command. For example, you can set the default principal as follows:

  ```
  asadmin set server-config.security-service.default-principal=dsmith
  asadmin set server-config.security-service.default-principal-password=secret
  ```

  You can set the default principal to role mapping as follows.

  ```
  asadmin set server-config.security-service.activate-default-principal-to-role-
  mapping=true
  asadmin set server-config.security-service.mapped-principal-class
  =CustomPrincipalImplClass
  ```

Default principal to role mapping is enabled by default. To disable it, set the default principal to role mapping property to `false`.

Realm Configuration

Supported Security Realms

The following security realms are supported:

- `file` - Stores user information in a file. This is the default realm.
- `ldap` - Stores user information in an LDAP directory.
- `jdbc` - Stores user information in a database.

  In the JDBC realm, the server gets user credentials from a database Payara Server uses the database information and the enabled JDBC realm option in the configuration file.

  For digest authentication, a JDBC realm should be created with `jdbcDigestRealm` as the JAAS context.

- `certificate` - Sets up the user identity in the Payara Server's security context, and populates it with user data obtained from cryptographically verified client certificates.
- `solaris` - Allows authentication using Solaris `username+password` data.

NOTE	This realm is only supported on the Solaris operating system, version 9 and above

How to Configure a Security Realm

You can configure a realm in one of these ways:

- In the Administration Console, open the *Security* component under the relevant configuration and go to the *Realms* page.
- Use the `asadmin create-auth-realm` command to create a new security realms.

How to Set a Realm for an Application or Module

The following deployment descriptor elements have optional `realm` or `realm-name` data sub-elements or attributes that override the domain's default realm:

- `glassfish-application`/`payara-application` element in `glassfish-application.xml`/`payara-application.xml`
- `web-app` element in `web.xml`
- `as-context` element in `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml`
- `client-container` element in `sun-acc.xml`
- `client-credential` element in `sun-acc.xml`

If modules within an application specify realms, these are ignored. If present, the realm defined in `glassfish-application.xml`/`payara-application.xml` is used, otherwise the domain's default realm is used.

For example, a realm is specified in `payara-application.xml` as follows:

```
<payara-application>
    ...
    <realm>ldap</realm>
</payara-application>
```

For more information about the deployment descriptor files and elements, see "elements-of-the-payara-platform-deployment-descriptors" in the Payara Server Application Deployment section.

Creating a Custom Realm

You can create a custom security realm by providing a custom Jakarta Authentication login module class and a custom security realm class.

CAUTION	Client-side login modules are not suitable for use with Payara Server.

To activate custom login modules and security realms, place their corresponding JAR files in the `domain-dir/lib` directory or the class files in the `domain-dir/lib/classes` directory.

For more information about class loading in the Payara Server, see class-loaders.

Jakarta Authentication is a set of APIs that enable services to authenticate and enforce access controls upon users. They provide a pluggable and extensible framework for programmatic user authentication and authorization.

For Javadoc tool pages relevant to custom realms, see the `com.sun.appserv.security` package.

Custom login modules must extend the `com.sun.appserv.security.AppservPasswordLoginModule` class. This class implements `jakarta.security.auth.spi.LoginModule`.

IMPORTANT	Custom login modules must not implement the `LoginModule` interface directly.

Custom login modules must provide an implementation for one `abstract` method defined in the `AppservPasswordLoginModule` class:

```
public abstract class AppservPasswordLoginModule extends BasePasswordLoginModule {
    abstract protected void authenticateUser() throws LoginException;
}
```

This method performs the actual authentication. The custom login module must not implement any of the other methods, such as `login`, `logout`, `abort`, `commit`, or `initialize`.

TIP	Default implementations are provided in `AppservPasswordLoginModule` which hook into the Payara Server infrastructure.

The custom login module can access the following protected object fields, which it inherits from `AppservPasswordLoginModule`. These contain the username and password credentials of the user to be authenticated:

```
public abstract class AppservPasswordLoginModule extends BasePasswordLoginModule {
    protected String _username;
    protected String _password;
}
```

The `authenticateUser` method must end with the following code sequence:

```
public abstract class AppservPasswordLoginModule extends BasePasswordLoginModule {

    protected void authenticateUser() throws LoginException{
        String[] grpList;
        // populate grpList with the set of groups to which
        // _username belongs in this realm, if any
        commitUserAuthentication(grpList);
    }
}
```

Custom security realms must extend the `com.sun.appserv.security.AppservRealm` class and implement the following methods:

```
public abstract class AppservRealm extends BaseRealm {
    public void init(Properties props) throws BadRealmException,
NoSuchRealmException;
}
```

This method is invoked during server startup when the realm is initially loaded. The `props` argument contains the configuration properties defined for this realm. The realm can do any initialization it needs in this method.

If the method returns without throwing an exception, Payara Server assumes that the realm is ready to service authentication requests. If an exception is thrown, the realm is disabled.

```
public abstract class AppservRealm extends BaseRealm {
    public String getAuthType();
}
```

This method returns a descriptive string representing the type of authentication done by this realm.

```
public abstract class AppservRealm extends BaseRealm {
    public abstract Enumeration getGroupNames(String username) throws
InvalidOperationException, NoSuchUserException;
}
```

This method returns an `Enumeration` (of `String` objects) enumerating the groups (if any) to which the given `username` belongs in this realm.

Custom realms that manage users must implement the following additional methods:

```
public abstract class AppservRealm extends BaseRealm {
    public abstract boolean supportsUserManagement();
}
```

This method returns `true` if the realm supports user management.

```
public abstract class AppservRealm extends BaseRealm {
    public abstract Enumeration getGroupNames() throws BadRealmException;
}
```

This method returns an `Enumeration` of all group names.

```
public abstract class AppservRealm extends BaseRealm {
    public abstract Enumeration getUserNames() throws BadRealmException;
}
```

This method returns an `Enumeration` of all usernames.

```
public abstract class AppservRealm extends BaseRealm {
    public abstract void refresh() throws BadRealmException;
}
```

This method refreshes the realm data so that new users and groups are visible.

```
public abstract class AppservRealm extends BaseRealm {
    public abstract void persist() throws BadRealmException;
}
```

This method persists the realm data to permanent storage.

```
public abstract class AppservRealm extends BaseRealm {
    public abstract User getUser(String name) throws NoSuchUserException
```

```
,BadRealmException;
}
```

This method returns the information recorded about a particular named user.

```
public abstract class AppservRealm extends BaseRealm {
    public abstract void addUser(String name, String password, String[] groupList)
throws BadRealmException, IASSecurityException;
}
```

This method adds a new user, who cannot already exist in the realm's storage.

```
public abstract class AppservRealm extends BaseRealm {
    public abstract void removeUser(String name) throws NoSuchUserException,
BadRealmException;
}
```

This method removes a user, who must exist in the realm's storage.

```
public abstract class AppservRealm extends BaseRealm {
    public abstract void updateUser(String name, String newName, String password,
String[] groups) throws NoSuchUserException, BadRealmException,
IASSecurityException;
}
```

This method updates data for a user, who must exist.

NOTE

The array passed to the `commitUseAuthentication` method should be newly created and otherwise unreferenced. This is because the group name array elements are set to null after authentication as part of cleanup.

So the second time your custom realm executes it returns an array with `null` elements.

Ideally, your custom realm should not return member variables from the `authenticate` method. It should return local variables Your custom realm can create a local `String` array in its `authenticate` method, copy the values from the member variables, and return the `String` array.

The `server.policy` File

Each Payara Server domain has its own global Java policy file, located in `domain-dir/config`. The file is named `server.policy`.

Payara Server is a Jakarta EE compliant application server. As such, it follows the requirements of the Jakarta EE specification, including the ability to run application components in the presence of a security manager (the internal Java component that enforces the policy) and a limited permission set for Jakarta EE application code.

Default Permissions

Internal server code is granted all permissions. These are covered by the `AllPermission` grant blocks to various parts of the server infrastructure code.

Application permissions are granted in the default `grant` block. These permissions apply to all code not part of the internal server code listed previously.Payara Server does not distinguish between EJB and web module permissions.

All code is granted the minimal set of web component permissions (which is a superset of the EJB minimal set).

A few permissions above the minimal set are also granted in the default `server.policy` file. These are necessary due to various internal dependencies of the server implementation. Jakarta EE application developers must not rely on these additional permissions.

In some cases, deleting these permissions might be appropriate. For example, one additional permission is granted specifically for using connectors. If connectors are not used in a particular domain, you should remove this permission, because it is not otherwise necessary.

CAUTION	Unless strictly required, do not modify the entries for the default permissions under any circumstance.

System Properties

The following predefined system properties, also called system variables, are available for use in the `server.policy` file.

NOTE The system property most frequently used in the `server.policy` file is `${com.sun.aas.instanceRoot}`.

Table 182. Predefined System Properties

Property	Default	Description
`com.sun.aas.installRoot`	Depends on operating system	Specifies the directory where Payara Server is installed.
`com.sun.aas.instanceRoot`	Depends on operating system	Specifies the top level directory for a server instance.
`com.sun.aas.hostName`	`none`	Specifies the name of the host (machine).
`com.sun.aas.javaRoot`	Depends on operating system	Specifies the installation directory for the Java runtime.
`com.sun.aas.imqLib`	Depends on operating system	Specifies the library directory for the Open Message Queue software.
`com.sun.aas.configName`	`server-config`	Specifies the name of the configuration used by a server instance.
`com.sun.aas.instanceName`	`server1`	Specifies the name of the server instance.
`com.sun.aas.clusterName`	`cluster1`	Specifies the name of the cluster. This property is only set on clustered server instances. **WARNING** This property is not supported for **deployment groups**.
`com.sun.aas.domainName`	`domain1`	Specifies the name of the domain.

WARNING	The `com.sun.aas.instanceName`, `com.sun.aas.clusterName` and `com.sun.aas.domainName` properties are not used in the default configuration, but can be used to customize configuration.

Changing Permissions for an Application

The default policy for each domain limits the permissions of Jakarta EE deployed applications to the minimal set of permissions required for these applications to operate correctly.

Do not add extra permissions to the default set (the grant block with no codebase, which applies to all code). Instead, add a new grant block with a codebase specific to the applications requiring the extra permissions, and only add the minimally necessary permissions in that block.

If you develop multiple applications that require more than this default set of permissions, you can add the custom permissions that your applications need. The `com.sun.aas.instanceRoot` variable refers to the domain directory. For example:

```
grant codeBase "file:${com.sun.aas.instanceRoot}/applications/-" {
...
}
```

You can add permissions to stub code with the following grant block:

```
grant codeBase "file:${com.sun.aas.instanceRoot}/generated/-" {
...
}
```

In general, you should add extra permissions only to the applications or modules that require them, not to all applications deployed to a domain. To apply permissions for a specific application, for example:

```
grant codeBase "file:${com.sun.aas.instanceRoot}/applications/MyApp/-" {
...
}
```

And for a specific module:

```
grant codeBase "file:${com.sun.aas.instanceRoot}/applications/MyModule/-" {
...
}
```

An alternative way to add permissions to a specific application or module is to edit the `granted.policy` file for that application or module. The `granted.policy` file is located in the `domain-dir/generated/policy/<app-or-module-name>` directory. In this case, you add permissions to the default `grant` block.

CAUTION	Do not delete permissions from this file.

When the Payara Server policy subsystem determines that a permission should not be granted, it logs a `server.policy` message specifying the permission that was not granted and the protection domains, with indicated code source and principals that failed the protection check.

For example, here is a similar entry:

```
[#|INFO|jakarta.enterprise.system.core.security|_ThreadID=14;_ThreadName=Thread-
31;|JACC Policy Provider: PolicyProviderImpl.implies, context(null)-
permission((java.util.PropertyPermission java.security.manager write)) domain that
failed(ProtectionDomain(file:/glassfish/domains/domain1/applications/sample/ ... )
```

Granting the following permission eliminates the message:

```
grant codeBase "file:${com.sun.aas.instanceRoot}/applications/sample/-" {
    permission java.util.PropertyPermission "java.security.manager", "write";
}
```

IMPORTANT	Do not add `java.security.AllPermission` to the `server.policy` file for application code. Doing so completely defeats the purpose of the security manager, yet you still get the performance overhead associated with it.

As a last resort, you can iteratively determine the permission set an application needs

by observing `AccessControlException` occurrences in the server's log.

If this is not sufficient, you can add the `-Djava.security.debug=failure` JVM option to the domain. Use the following `asadmin create-jvm-options` command, then restart the server:

```
asadmin create-jvm-options -Djava.security.debug=failure
```

Enabling and Disabling the Security Manager

IMPORTANT | The security manager is disabled by default.

In a production environment, you may be able to safely disable the security manager if all the following conditions are true:

- Performance is critical
- Deployment to the production server is carefully controlled
- Only trusted applications are deployed
- Applications don't need a strict policy enforcement

Disabling the security manager may improve performance significantly for some types of applications.

To enable the security manager, do one of the following:

- To use the Administration Console, open the *Security* component under the relevant configuration, and check the *Security Manager Enabled* box. Then restart the server instance.
- Use the following `asadmin create-jvm-options` command, then restart the server:

  ```
  asadmin create-jvm-options -Djava.security.manager
  ```

To disable the security manager, uncheck the *Security Manager Enabled* box or use the

corresponding `asadmin delete-jvm-options` command.

You must grant additional permissions to CDI-enabled Jakarta EE applications that are deployed in a Payara Server domain, cluster or deployment group for which the security manager is enabled. These additional permissions are not required when it is disabled.

To deploy CDI-enabled Jakarta EE applications in this scenario in which security manager is enabled, add the following permissions to the corresponding applications:

```
grant codeBase "file:${com.sun.aas.instanceRoot}/applications/[ApplicationName]" {
    permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
};
```

For example, for a CDI application named `foo.war`, add the following permissions to the `server.policy` file, restart the domain, cluster or deployment group, and then deploy and use the application.

```
grant codeBase "file:${com.sun.aas.instanceRoot}/applications/foo" {
    permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
};
```

Configuring Message Security for Web Services

In message security, security information is applied at the message layer and travels along with the web services message. Web Services Security (WSS) is the use of XML Encryption and XML Digital Signatures to secure messages.

WSS profiles the use of various security tokens including X.509 certificates, Security Assertion Markup Language (SAML) assertions, and username/password tokens to achieve this.

Message layer security differs from transport layer security in that it can be used to decouple message protection from message transport so that messages remain protected after transmission, regardless of how many hops they travel.

IMPORTANT	Message layer annotations are not currently supported.

For more information about web services, see developing-web-services.

For more information about message security, see the following:

- Introduction to Security in the Jakarta EE Platform in The Jakarta EE Tutorial
- Jakarta Authorization
- The Oasis Web Services Security (WSS) specification
- The Web Services Interoperability Organization (WS-I) Basic Security Profile (BSP) specification
- The Metro User's guide WSIT Security documentation.

Message Security Providers

By default, the `XWS_ClientProvider` and `XWS_ServerProvider` security message providers are configured but disabled. You can enable them in one of the following ways:

- To enable the message security providers using the Administration Console, open the *Security* component under the relevant configuration, select the *Message Security* component, and select *SOAP*. Then select `XWS_ServerProvider` from the *Default Provider* list and `XWS_ClientProvider` from the *Default Client Provider* list.
- You can enable the message security providers using the following Asadmin CLI commands:

```
asadmin set
server-config.security-service.message-security-config.SOAP.default_provider
=XWS_ServerProvider
asadmin set
server-config.security-service.message-security-
config.SOAP.default_client_provider=XWS_ClientProvider
```

You can create new message security providers in one of the following ways:

- To create a message security provider using the Administration Console, open the *Security* component under the relevant configuration, and select the *Message Security* component.

- You can use the `asadmin create-message-security-provider` command to create a message security provider.

In addition, you can set a few optional provider properties using the `asadmin set` command. For example:

```
asadmin set server-config.security-service.message-security-config.provider-
config.property.debug=true
```

The following table describes these message security provider properties.

Table 183. Message Security Provider Properties

Property	Default	Description
`security.config`	`domain-dir/config/wss-server-config-1.0.xml`	Specifies the location of the message security configuration file. To point to a configuration file in the `domain-dir/config` directory, use the system property like this: `${com.sun.aas.instanceRoot}/config/wss-server-config-1.0.xml`
`debug`	`false`	If `true`, enables dumping of server provider debug messages to the server log.
`dynamic.username.password`	`false`	If `true`, signals the provider runtime to collect the username and password from the `CallbackHandler` for each request. If `false`, the username and password for `wsse:UsernameToken(s)` is collected once, during module initialization. This property is only applicable for a `ClientAuthModule`.

Property	Default	Description
`encryption.key.alias`	`s1as`	Specifies the encryption key used by the provider. The key is identified by its `keystore` alias.
`signature.key.alias`	`s1as`	Specifies the signature key used by the provider. The key is identified by its `keystore` alias.

Message Security Responsibilities

In Payara Server, the system administrator and application deployer roles are expected to take primary responsibility for configuring message security.

In some situations, the application developer may also contribute, although in the typical case either of the other roles may secure an existing application without changing its implementation and without involving the developer.

Application Developer Responsibilities

The application developer can turn on message security, but is not responsible for doing so. Message security can be set up by the system administrator so that all web services are secured, or set up by the application deployer when the provider or protection policy bound to the application must be different from that bound to the container.

The application developer is responsible for the following:

- Determining if an application-specific message protection policy is required by the application. If so, ensuring that the required policy is specified at application assembly which may be accomplished by communicating with the application deployer.
- Determining if message security is necessary at the server's level. If so, ensuring that this need is communicated to the system administrator, or taking care of implementing message security at the server's level.

Application Deployer Responsibilities

The application deployer is responsible for the following:

- Specifying (at application assembly) any required application-specific message protection policies if such policies have not already been specified by upstream roles (the developer or assembler).
- Modifying Payara Server deployment descriptors to specify application-specific message protection policies information (message-security-binding elements) to web service endpoint and service references

These security tasks are discussed in Application-Specific Message Protection.

System Administrator Responsibilities

The system administrator is responsible for the following:

- Configuring message security providers on Payara Server.
- Managing user databases.
- Managing keystore and truststore files.

A system administrator uses the Administration Console to manage server security settings and uses a command line tool to manage certificate databases. Certificates and private keys are stored in key stores and are managed with the `keytool` utility (and in the case of Payara Server Enterprise with its Integrated Certificate Management enhancements).

Application-Specific Message Protection

When the Payara Server provided configuration is insufficient for your security needs, and you want to override the default protection, you can apply application-specific message security to a web service.

Application-specific security is implemented by adding the message security binding to the web service endpoint, whether it is an EJB or servlet web service endpoint. Modify the corresponding XML deployment descriptor files to add the message binding

information.

Message security can also be specified using a WSIT security policy in the WSDL file.

For more information about message security providers, see Message Security Providers.

For more details on message security binding for EJB web services,servlet web services, and clients, see the XML file descriptions in "elements-of-the-payara-platform-deployment-descriptors" in the Payara Server Application Deployment section.

Using a Signature to Enable Message Protection for All Methods

To enable message protection for all methods using digital signature, update the `message-security-binding` element for the EJB web service endpoint in the application's `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` file.

To apply the same protection mechanisms for all methods, leave the method-name element blank. Configuring Message Protection for a Specific Method Based on Digital Signatures discusses listing specific methods or using wildcard characters.

To Enable Message Protection for All Methods Using Digital Signature

Follow this procedure:

1. Modify the `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` file by adding the `message-security-binding` element as shown:

```
<payara-ejb-jar>
  <enterprise-beans>
    <unique-id>1</unique-id>
    <ejb>
      <ejb-name>HelloWorld</ejb-name>
      <jndi-name>HelloWorld</jndi-name>
      <webservice-endpoint>
        <port-component-name>HelloIF</port-component-name>
        <endpoint-address-uri>service/HelloWorld</endpoint-address-uri>
        <message-security-binding auth-layer="SOAP">
          <message-security>
            <request-protection auth-source="content" />
            <response-protection auth-source="content"/>
          </message-security>
        </message-security-binding>
      </webservice-endpoint>
```

```
    </ejb>
  </enterprise-beans>
</payara-ejb-jar>
```

2. Compile and deploy the corresponding application.

Configuring Message Protection for a Specific Method Based on Digital Signatures

To enable message protection for a specific method, or for a set of methods that can be identified using a wildcard value, follow these steps.

As in the example discussed in Using a Signature to Enable Message Protection for All Methods, to enable message protection for a specific method, update the `message-security-binding` element for the EJB web service endpoint in the application's `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` file.

Add the `request-protection` and `response-protection` elements, which are analogous to the `request-policy` and `response-policy` elements discussed in the Azul Payara Security Guide.

The administration guide includes a table listing the set and order of security operations for different request and response policy configurations.

To Enable Message Protection for a Particular Method or Set of Methods Using Digital Signature

Follow this procedure.

1. Modify the `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` file by adding the `message-security-binding` element as shown:

```
<payara-ejb-jar>
  <enterprise-beans>
  <unique-id>1</unique-id>
    <ejb>
      <ejb-name>HelloWorld</ejb-name>
      <jndi-name>HelloWorld</jndi-name>
      <webservice-endpoint>
        <port-component-name>HelloIF</port-component-name>
        <endpoint-address-uri>service/HelloWorld</endpoint-address-uri>
        <message-security-binding auth-layer="SOAP">
          <message-security>
            <message>
```

```
            <java-method>
              <method-name>ejbCreate</method-name>
            </java-method>
          </message>
          <message>
            <java-method>
              <method-name>sayHello</method-name>
            </java-method>
          </message>
          <request-protection auth-source="content" />
          <response-protection auth-source="content"/>
        </message-security>
      </message-security-binding>
    </webservice-endpoint>
  </ejb>
 </enterprise-beans>
</payara-ejb-jar>
```

2. Compile and deploy the corresponding application.

Programmatic Login Using the Programmaticlogin Class

Programmatic login allows a deployed Jakarta EE application or module to invoke a **login** method. If the login is successful, a `SecurityContext` is established as if the client had authenticated using any of the conventional Jakarta EE mechanisms.

Programmatic login is supported for servlet and EJB components on the server side, and for stand-alone or application clients on the client side.

Programmatic login is useful for an application having special needs that cannot be accommodated by any of the Jakarta EE standard authentication mechanisms.

This section describes a proprietary Payara mechanism, but see also the standard security APIs in the Jakarta EE tutorial.

NOTE

The `com.sun.enterprise.security.ee.auth.login.ProgrammaticLogin` class present in Payara Server is not part of any Jakarta EE APIs; therefore, it is not portable to other application servers.

Programmatic Login Precautions

Payara Server is not involved in how the login information (`user`, `password`) is obtained by the deployed application. Programmatic login places the burden on the

application developer with respect to assuring that the resulting system meets security requirements. If the application code reads the authentication information across the network, the application determines whether to trust the user.

Programmatic login allows the application developer to bypass the Payara Server-supported authentication mechanisms and feed authentication data directly to the security service.

CAUTION	While flexible, this capability should not be used without some understanding of security issues. Since this mechanism bypasses the container-managed authentication process and sequence, the application developer must be very careful in making sure that authentication is established before accessing any restricted resources or methods.

It is also the application developer's responsibility to verify the status of the login attempt and to alter the behavior of the application accordingly.

The programmatic login state does not necessarily persist in sessions or participate in single sign-on.

Lazy authentication is not supported for programmatic login. If an access check is reached and the deployed application has not properly authenticated using the programmatic login method, access is denied immediately and the application might fail if not coded to account for this occurrence.

One way to account for this occurrence is to catch the access control or security exception, perform a programmatic login, and repeat the request.

Granting Programmatic Login Permission

The `ProgrammaticLoginPermission` permission is required to invoke the programmatic login mechanism for an application if the security manager is enabled. For information about the security manager, see The `server.policy` File.

This permission is not granted by default to deployed applications because this is not a standard Jakarta EE mechanism.

To grant the required permission to the application, add the following to the `domain-dir/config/server.policy` file:

```
grant codeBase "file:${jar-file-path}" {
    permission com.sun.appserv.security.ProgrammaticLoginPermission "login";
};
```

NOTE | `jar-file-path` is the path to the application's JAR file.

The `ProgrammaticLogin` Class

The `com.sun.enterprise.security.ee.auth.login.ProgrammaticLogin` class enables a user to perform login programmatically and in addition, logout of the existing security context when required.

The `ProgrammaticLogin` class has eight overloaded `login` methods: four variants to be used by Web components and four variants to be used by EJB components.

The login methods for web components have the following signatures:

```
public class ProgrammaticLogin {
    public Boolean login(String user, String password, HttpServletRequest request,
HttpServletResponse response);

    //Same as above, but uses a char[] for the password credential
    public Boolean login(String user, char[] password, HttpServletRequest request,
HttpServletResponse response);

    public Boolean login(String user, String password, String realm,
HttpServletRequest request, HttpServletResponse response, boolean errors) throws
Exception;

    //Same as above, but uses a char[] for the password credential
    public Boolean login(String user, char[] password, String realm,
HttpServletRequest request, HttpServletResponse response, boolean errors) throws
Exception;
}
```

The login methods for EJB components have the following signatures:

```
public class ProgrammaticLogin {
```

```
    public Boolean login(String user, String password);
    //Same as above, but uses a char[] for the password credential
    public Boolean login(String user, char[] password);
    public Boolean login(String user, String password, String realm, boolean
errors) throws Exception;
    //Same as above, but uses a char[] for the password credential
    public Boolean login(String user, char[] password, String realm, boolean
errors);
}
```

All method variants accomplish the following:

- Perform the authentication
- Return `true` if login succeeded, `false` if the login has failed

The login occurs on the realm specified unless it is `null`, in which case the domain's default realm is used. The variants with no `realm` parameter will use the domain's default security realm.

If the `errors` parameter is set to `true`, any exceptions encountered during the login process are propagated to the caller. If set to `false`, exceptions will be thrown.

WARNING

On the client side, the `realm` and `errors` parameters are ignored and the actual login does not occur until a resource requiring a login is accessed. A `java.rmi.AccessException` with a `COBRA NO_PERMISSION` occurs if the actual login fails.

The `logout` method has 4 overloaded variants: 2 used by Web components, 2 used by EJB components. Here are the variants used by web components:

```
public class ProgrammaticLogin {
    public Boolean logout(HttpServletRequest request, HttpServletResponse
response);
    public Boolean logout(HttpServletRequest request, HttpServletResponse response,
boolean errors) throws Exception;
}
```

The variants used for EJB components have the following signatures:

```
public class ProgrammaticLogin {
    public Boolean logout();
```

```
    public Boolean logout(boolean errors) throws Exception;
}
```

All `logout` method variants return `true` if the logout process succeeded, `false` if it failed.

If the `errors` parameter is set to `true`, any exceptions encountered during the logout process are propagated to the caller. If set to `false`, exceptions will be thrown.

User Authentication for Single Sign-On

The single sign-on feature of the Payara Server allows multiple web applications deployed to the same virtual server to share the user authentication state.

With single sign-on enabled, users who log in to one web application become **implicitly logged** into other web applications on the same virtual server that require the same authentication information. Otherwise, users would have to log in separately to each web application whose protected resources they tried to access.

A sample application using the single sign-on scenario could be a consolidated airline booking service that searches all airlines and provides links to different airline websites. After the user signs on to the consolidated booking service, the user information can be used by each individual airline site without requiring another sign-on.

Single sign-on operates according to the following rules:

- Single sign-on applies to web applications configured for the same security realm and virtual server. The realm is defined by the `realm-name` element in the `web.xml` file. For information about virtual servers, see "administering-http-connectivity" in the Payara Server General Administration section.
- As long as users access only unprotected resources in any of the web applications on a virtual server, they are not challenged to authenticate themselves.
- As soon as a user accesses a protected resource in any web application associated with a virtual server, the user is challenged to authenticate himself or herself, using the login method defined for the web application currently being accessed.

- After authentication, the roles associated with this user are used for access control decisions across all associated web applications, without challenging the user to authenticate to each application individually.
- When the user logs out of one web application (for example, by invalidating the corresponding web session), the user's sessions in all web applications are automatically invalidated. Any subsequent attempt to access a protected resource in any application requires the user to authenticate again.

IMPORTANT	The single sign-on feature utilizes HTTP cookies to transmit a token that associates each request with the saved user identity, so it can only be used in client environments that support cookies.

To configure single sign-on, set the following configuration properties for a **virtual server**:

`sso-enabled`: If `false`, single sign-on is disabled for this virtual server, and users must authenticate separately to every application on the virtual server. The default value is `false`.

`sso-max-inactive-seconds`: Specifies the time after which a user's single sign-on record becomes eligible for purging if no client activity is received. Since single sign-on applies across several applications on the same virtual server, access to any of the applications keeps the single sign-on record active. The default value is *5 minutes* (`300` seconds).

+ CAUTION: Higher values provide longer single sign-on persistence for the users at the expense of more memory use on the server.

`sso-reap-interval-seconds`: Specifies the interval between purges of expired single sign-on records. The default value is `60`.

Here are example `asadmin set` commands on a virtual server named `vsrv1` using the default values mentioned above:

```
asadmin set server-config.http-service.virtual-server.vsrv1.property.sso-enabled
="true"
asadmin set server-config.http-service.virtual-server.vsrv1.property.sso-max-
inactive-seconds="300"
asadmin set server-config.http-service.virtual-server.vsrv1.property.sso-reap-
interval-seconds="60"
```

Jakarta Security Support

Jakarta Security defines several authentication-related plugin SPIs, such as the `HttpAuthenticationMechanism` and `IdentityStore` interfaces:

`HttpAuthenticationMechanism`: An interface for modules that authenticate callers to a web application. An application can supply its own `HttpAuthenticationMechanism`, or use one of the default implementations provided by the container. **`IdentityStore`**: This interface defines methods for validating a caller's credentials (such as username and password) and returning group membership information. An application can provide its own `IdentityStore`, or use the builtin LDAP or Database stores.

In addition to these authentication plugin SPIs, the Jakarta Security specification defines the `SecurityContext` API for use by application code to query and interact with the current security context.

The `SecurityContext` interface defines methods that allow an application to access security information about a caller, authenticate a caller, and authorize a caller. These methods include `getCallerPrincipal()`, `getPrincipalsByType()`, `isCallerInRole()`, `authenticate()`, and `hasAccessToWebResource()`.

Azul Payara and Jakarta Security

Payara Server and Micro implement Jakarta Security to provide built-in support for the following authentication mechanisms:

- BASIC
- FORM
- Custom FORM

- OpenID Connect

A significant advantage of using `HttpAuthenticationMechanism` and `IdentityStore` over the declarative mechanisms defined by the Servlet specification is that it allows an application to control the identity stores that it authenticates against, in a standard, portable way. You can use the built-in implementations of these APIs, or define custom implementations as suggested by the specification.

Jakarta Security defines several annotations, with names that end with the `Definition` suffix, which when used makes the corresponding built-in mechanism available as a CDI bean.

TIP	Jakarta Security also supports the use of Jakarta Expression Language in these annotations to allow dynamic configuration.

Built-In Authentication Mechanisms and Identity Stores

An application packages its own `HttpAuthenticationMechanism` by including in a bean archive that is a part of the application. Alternatively, it may select and configure one of the built-in mechanisms using the corresponding annotation, as listed below:

`BasicAuthenticationMechanismDefinition`: Implements HTTP BASIC authentication that conforms to the behavior of the servlet container when BASIC <auth-method> is declared in the `web.xml` deployment descriptor.
`FormAuthenticationMechanismDefinition`: Implements HTTP FORM authentication that conforms to the behavior of the servlet container when the FORM <auth-method> is declared in the `web.xml` deployment descriptor.
`CustomFormAuthenticationMechanismDefinition`: Implements a modified version of HTTP FORM authentication in which custom handling replaces the POST action to the `j_security_check` servlet.
`OpenIdAuthenticationMechanismDefinition`: Implements an OpenID Connect authentication workflow using the metadata configured in the annotation attributes.

All built-in authentication mechanisms need to be authenticated using an identity store.

The `IdentityStore` interface, included in the Jakarta Security API, defines an SPI for interacting with identity stores, which are repositories (like databases or naming directories) containing user account information.

The `IdentityStore` interface has four methods: `validate(Credential)`, `getCallerGroups(CredentialValidationResult)`, `validationTypes()` and `priority()`. Developers can provide their own implementation of this interface, or use one of the built-in identity stores.

> **TIP**
>
> The `RememberMeIdentityStore` interface, which is a variation on the `IdentityStore` interface, can be used when an application wants to "remember" a user's authenticated session for an extended period, so that the caller can return to the application periodically without needing to present primary authentication credentials each time.

There are two built-in implementations of `IdentityStore`:

`LdapIdentityStoreDefinition`: Used to configure an LDAP naming directory as the identity store of an application. **`DatabaseIdentityStoreDefinition`**: Used to configure a set of relational database tables as the identity store.

The following snippet shows the usage of the `DatabaseIdentityStoreDefinition` annotation, which makes it available as a CDI bean.

```
@DatabaseIdentityStoreDefinition(
    callerQuery = "#{'select password from caller where name = ?'}",
    groupsQuery = "select group_name from caller_groups where caller_name = ?",
    hashAlgorithm = Pbkdf2PasswordHash.class,
    priorityExpression = "#{100}",
    hashAlgorithmParameters = {
        "Pbkdf2PasswordHash.Iterations=3072",
        "${applicationConfig.dynamic}"
    }
)
public class MyComponent{
}
```

Since the Jakarta Security API provides support for Jakarta Expression Language, regular expressions can be used to set value of annotation attributes.

Payara Server provides out of the box implementation of the `Pbkdf2PasswordHash` interface that supports PBKDF2 password hashing.

It is suggested that you use `Pbkdf2PasswordHash` for generating and validating passwords, unless there are specific requirements which cannot be met in any other way.

Custom Authentication Mechanism

An application provider can choose to provide its own custom authentication mechanism, apart from built-in authentication mechanisms.

A custom authentication mechanism implements the `HttpAuthenticationMechanism` interface, introduced in Jakarta Security. This interface defines the following three methods.

```
public interface HttpAuthenticationMechanism{
    AuthenticationStatus validateRequest(HttpServletRequest request,
HttpServletResponse response, HttpMessageContext httpMessageContext) throws
AuthenticationException;

    AuthenticationStatus secureResponse(HttpServletRequest request,
HttpServletResponse response, HttpMessageContext httpMessageContext) throws
AuthenticationException;

    void cleanSubject(HttpServletRequest request, HttpServletResponse response,
HttpMessageContext httpMessageContext);
}
```

A custom authentication mechanism must return an `AuthenticationStatus` to indicate the status of its current authentication request. Internally, it gets translated to a corresponding Jakarta Authentication `AuthStatus` instance as shown below:

- `AuthenticationStatus.NOT_DONE` → `AuthStatus.SUCCESS`
- `AuthenticationStatus.SEND_CONTINUE` → `AuthStatus.SEND_CONTINUE`
- `AuthenticationStatus.SUCCESS` → `AuthStatus.SUCCESS`
- `AuthenticationStatus.SEND_FAILURE` → `AuthStatus.SEND_FAILURE`

Each method of the `HttpAuthenticationMechanism` interface performs the same

function as the corresponding `ServerAuth` methods.

Unlike Jakarta Authentication, `HttpAuthenticationMechanism` is specified for the servlet container only. Only the `validateRequest()` must be implemented, for other two methods, default behaviors are specified.

`validateRequest` allows a caller to authenticate. The request gets inspected inside `validateRequest` to read credential or any other information, or it can write to standard response with status of the authentication request or redirect the caller to an OAuth provider. Once the credential is validated, the result of the validation is communicated to the container using the `HttpMessageContext` parameter.

Here's an example of a custom authentication mechanism:

NOTE

Only the `validateRequest` method has been implemented, since Jakarta Security provides default implementation of the other two methods. An application provider may choose to override the default implementation depending on the business context.

```
import jakarta.enterprise.context.RequestScoped;
import jakarta.inject.Inject;
import jakarta.security.enterprise.AuthenticationException;
import jakarta.security.enterprise.AuthenticationStatus;
import
jakarta.security.enterprise.authentication.mechanism.http.HttpAuthenticationMechani
sm;
import
jakarta.security.enterprise.authentication.mechanism.http.HttpMessageContext;
import jakarta.security.enterprise.credential.UsernamePasswordCredential;
import jakarta.security.enterprise.identitystore.CredentialValidationResult;
import jakarta.security.enterprise.identitystore.IdentityStoreHandler;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import static jakarta.security.enterprise.identitystore.CredentialValidationResult
.Status.VALID;

@RequestScoped
public class MyAuthenticationMechanism implements HttpAuthenticationMechanism {

    @Inject
    private IdentityStoreHandler identityStoreHandler;

    @Override
    public AuthenticationStatus validateRequest(HttpServletRequest request,
```

```
HttpServletResponse response, HttpMessageContext httpMessageContext) throws
AuthenticationException {

        // Get the (caller) name and password from the request
        // NOTE: This is for the smallest possible example only. In practice
        // putting the password in a request query parameter is highly
        // insecure and is discouraged.
        var name = request.getParameter("name");
        var password = request.getParameter("password");

        if (name != null && password != null) {
            // Delegate the {credentials in -> identity data out} function to the
Identity Store
            var result = identityStoreHandler.validate(new
UsernamePasswordCredential(name, password));

            if (result.getStatus() == VALID) {
                // Communicate the details of the authenticated user to the
container.
                response.addHeader("Authentication Mechanism",
"MyAuthenticationMechanism");
                return httpMessageContext.notifyContainerAboutLogin(result
.getCallerPrincipal(), result.getCallerGroups());
            } else {
                return httpMessageContext.responseUnauthorized();
            }
        }
        return httpMessageContext.doNothing();
    }
}
```

Multiple Httpauthenticationmechanisms in EAR Application

Within the Security API specification, an `HttpAuthenticationMechanism` can be defined to retrieve the user credentials from the HTTP Request. These credentials are then used by any configured identity stores to validate the user.

However, a restriction of the Security API specification is that only one `HttpAuthenticationMechanism` can be defined per application.

This restriction is also in place for EAR applications file where it's possible to have multiple web modules.

Payara Server allows to circumvent this restriction so that multiple authentication mechanisms are used in an EAR application that hosts more than one Web module. To this effect, each web module must configure its own authentication mechanism separately.

IMPORTANT	In order to use this feature, there must not be any custom `HttpAuthenticationMechanism` implementations bundled in any of the modules hosted in the EAR artifact. If this is the case, the feature will not work as intended.

NOTE	This behaviour is specific to Azul Payara and not compatible with the Jakarta Security specification.

How to Configure

In order to configure a specific `HttpAuthenticationMechanism` for a Web application, define the `fish.payara.security.mechanism` parameter in the `web.xml` deployment descriptor like in the following case:

```
<context-param>
    <param-name>fish.payara.security.mechanism</param-name>
    <param-value>JWT</param-value>
</context-param>
```

The following mechanisms can be used:

Value	Description
`Basic`	Use *BasicAuthenticationMechanism* as mechanism.
`Form`	Use *FormAuthenticationMechanism* as mechanism.
`CustomForm`	Use *CustomFormAuthenticationMechanism* as mechanism.
`JWT`	Use the custom Payara *JWTAuthenticationMechanism* as mechanism.
`Certificate`	Use the custom Payara *CertificateAuthenticationMechanism* as mechanism.
`Azure`	Use the custom Payara *AzureOpenIdAuthenticationMechanism* as mechanism.
`Google`	Use the custom Payara *GoogleOpenIdAuthenticationMechanism* as mechanism.
`OAuth2`	Use the custom Payara *OAuth2AuthenticationMechanism* as mechanism.
<any Fully Qualified Class Name>	Use the `HttpAuthenticationMechanism` indicated by the class name.

NOTE	When no parameter is specified, the standard rules of Jakarta Security are active (only 1 mechanism allowed per application/module).

Jakarta Authentication Support

Jakarta Authentication is part of the Jakarta EE specification and defined by its own specification.

Jakarta Authentication defines an interface for pluggable authorization providers. Specifically, it is used to plug in the Java policy provider used by the container to perform Jakarta EE caller access decisions. The Java policy provider performs Java policy decisions during application execution.

This provides third parties with a mechanism to develop and plug in modules that are responsible for answering authorization decisions during a Jakarta EE application execution. The interfaces and rules used for developing authentication providers are defined in the specification document.

Payara Server provides a simple file-based Jakarta Authentication engine as a default **JACC provider** (JACC stands for the previous name of the specification before its transition to Jakarta EE), named `default`.

To configure an alternate provider using the Administration Console, open the *Security* component under the relevant configuration, and select the *JACC Providers* component.

Payara Server and Jakarta Authentication

The Jakarta Authentication specification defines profiles that establish contracts for the use of the SPI in specific contexts.

Jakarta Authentication's Servlet Container Profile defines the use of the SPI by a Servlet container such that:

- The resulting container can be configured with new authentication mechanisms.
- The container employs the configured mechanisms in its enforcement of the declarative servlet security model (declared in a `web.xml` deployment descriptor file

using `security-constraint` elements or by specific annotations in the servlet implementation classes).

Jakarta Authentication defines a simple message processing model composed of four interaction points when processing an authentication request:

1. `secureRequest` on the client
2. `validateRequest` on the server
3. `secureResponse` on the server
4. `validateResponse` on the client

A message processing runtime uses the SPI at these interaction points to delegate the corresponding message security processing to authentication providers, also called authentication modules, integrated into the runtime by way of the SPI.

A compatible server-side message processing runtime, such as the Payara Server servlet container, supports the `validateRequest` and `secureResponse` interaction points of the message processing model.

The servlet container uses the SPI at these interaction points to delegate the corresponding message security processing to a **server authentication module (SAM)**, integrated by the SPI into the container.

Adding Authentication Modules to the Servlet Container

You can use Jakarta Authentication in the web tier to facilitate the injection of pluggable authentication modules within the servlet constraint processing engine. Payara Server includes implementations of a number of HTTP layer authentication mechanisms such as `basic`, `form`, and `digest` authentication.

You can add alternative implementations of the included mechanisms or implementations of new mechanisms such as Kerberos/SPNEGO or CAS for example.

Writing a Server Authentication Module

A key step in adding an authentication mechanism to a compatible server-side message

processing runtime such as the servlet container is acquiring a SAM that implements the desired authentication mechanism. One way to do that is to write the SAM yourself.

A SAM implements the `jakarta.security.auth.message.module.ServerAuthModule` interface as defined by the specification. A SAM is invoked indirectly by the message processing runtime at the `validateRequest` and `secureResponse` interaction points. A SAM must implement the five methods of the `ServerAuthModule` interface:

`getSupportedMessageTypes`: Returns an array of `Class` objects where each element defines a message type supported by the SAM.

+ NOTE: For a SAM to be compatible with the Servlet Container Profile, the returned array must include the `HttpServletRequest.class` and `HttpServletResponse.class` objects.

`initialize(MessagePolicy requestPolicy, MessagePolicy responsePolicy, CallbackHandler cbh, Map options)`: The container calls this method to provide the SAM with configuration values and with a `CallbackHandler`.

+ The configuration values are returned in the policy arguments and in the options `Map`. The SAM uses `CallbackHandler` to access services, such as password validation, provided by the container.

`AuthStatus validateRequest(MessageInfo messageInfo, Subject clientSubject, Subject serviceSubject)`: The container calls this method to process each `HttpServletRequest` received. The request and its associated `HttpServletResponse` are passed by the container to the SAM in the `messageInfo` argument. The SAM processes the request and may establish the response to be returned by the container.

+ The SAM uses the provided `Subject` arguments to convey its authentication results. The SAM returns different status values to control the container's invocation processing. The status values and the circumstances under which they are returned are

as follows:

- `AuthStatus.SUCCESS` is returned when the application request message is successfully validated. The container responds to this status value by using the returned client `Subject` to invoke the target of the request. When this value is returned, the SAM (provided a custom `AuthConfigProvider` is not being used) must use its `CallbackHandler` to handle a `CallerPrincipalCallback` using the `clientSubject` as an argument to the callback.
- `AuthStatus.SEND_CONTINUE` indicates that message validation is incomplete and that the SAM has established a preliminary response as the response message in `messageInfo`. The container responds to this status value by sending the response to the client.
- `AuthStatus.SEND_FAILURE` indicates that message validation failed and that the SAM has established an appropriate failure response message in `messageInfo`. The container responds to this status value by sending the response to the client.
- `AuthStatus.SEND_SUCCESS` is not typically returned. This status value indicates the end of a multi-message security dialog originating after the service interaction and during the processing of the application response. The container responds to this status value by sending the response to the client.

  The `validateRequest` method may also throw an `AuthException` to indicate that the message processing by the SAM failed without establishing a failure response message in `messageInfo`.

`secureResponse(MessageInfo messageInfo, Subject serviceSubject)`: The container calls this method before sending a response, resulting from an application invocation, to the client. The response is passed to the SAM in the `messageInfo` argument. In most cases, this method should just return the `SEND_SUCCESS` status.

`cleanSubject(MessageInfo messageInfo, Subject subject)`: This method removes the mechanism-specific principals, credentials, or both from the subject. This

method is not currently called by the container. A legitimate implementation could remove all the principals from the argument subject.

Sample Server Authentication Module

The class `MySam.java` is a sample SAM implementation. Notice that the sample implements the five methods of the ServerAuthModule interface. This SAM implements an emulation of HTTP basic authentication for demonstration purposes.

```
package fish.payara.samples.security;

import java.io.IOException;
import java.util.Map;
import jakarta.security.auth.Subject;
import jakarta.security.auth.callback.Callback;
import jakarta.security.auth.callback.CallbackHandler;
import jakarta.security.auth.callback.UnsupportedCallbackException;
import jakarta.security.auth.message.AuthException;
import jakarta.security.auth.message.AuthStatus;
import jakarta.security.auth.message.MessageInfo;
import jakarta.security.auth.message.MessagePolicy;
import jakarta.security.auth.message.callback.CallerPrincipalCallback;
import jakarta.security.auth.message.callback.GroupPrincipalCallback;
import jakarta.security.auth.message.callback.PasswordValidationCallback;
import jakarta.security.auth.message.module.ServerAuthModule;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.apache.catalina.util.Base64;

public class MySam implements ServerAuthModule {

    protected static final Class[] supportedMessageTypes = new Class[]{
            HttpServletRequest.class, HttpServletResponse.class
    };

    private MessagePolicy requestPolicy;
    private MessagePolicy responsePolicy;
    private CallbackHandler handler;
    private Map options;
    private String realmName = null;
    private String defaultGroup[] = null;

    private static final String REALM_PROPERTY_NAME = "realm.name";
    private static final String GROUP_PROPERTY_NAME = "group.name";
    private static final String BASIC = "Basic";
    private static final String AUTH_TYPE_INFO_KEY =
"jakarta.servlet.http.authType";

    static final String AUTHORIZATION_HEADER = "authorization";
    static final String AUTHENTICATION_HEADER = "WWW-Authenticate";

    public void initialize(MessagePolicy requestPolicy, MessagePolicy
responsePolicy, CallbackHandler handler, Map options) throws AuthException {
        if (options != null) {
            realmName = (String) options.get(REALM_PROPERTY_NAME);
```

```
            if (options.containsKey(GROUP_PROPERTY_NAME)) {
              defaultGroup = new String[]{(String) options.get(
GROUP_PROPERTY_NAME)};
            }
        }
    }

    public Class[] getSupportedMessageTypes() {
      return supportedMessageTypes;
    }

    public AuthStatus validateRequest(MessageInfo msgInfo, Subject client, Subject
server) throws AuthException {
        try {
          var username = processAuthorizationToken(msgInfo, client);
          if (username == null && requestPolicy.isMandatory()) {
              return sendAuthenticateChallenge(msgInfo);
          }
          setAuthenticationResult(username, client, msgInfo);
          return AuthStatus.SUCCESS;
        } catch (Exception e) {
          var ae = new AuthException();
          ae.initCause(e);
          throw ae;
        }
    }

    private String processAuthorizationToken(MessageInfo msgInfo, Subject s) throws
AuthException {

        var request = (HttpServletRequest) msgInfo.getRequestMessage();
        var token = request.getHeader(AUTHORIZATION_HEADER);

        if (token != null && token.startsWith(BASIC + " ")) {
            token = token.substring(6).trim();

            // Decode and parse the authorization token
            var decoded = new String(Base64.decode(token.getBytes()));
            int colon = decoded.indexOf(':');
            if (colon <= 0 || colon == decoded.length() - 1) {
                return null;
            }

            var username = decoded.substring(0, colon);
            // use the callback to ask the container to validate the password
            var pVC = new PasswordValidationCallback(s, username, decoded.
substring(colon + 1).toCharArray());
            try {
                handler.handle(new Callback[]{pVC});
                pVC.clearPassword();
            } catch (Exception e) {
                var ae = new AuthException();
                ae.initCause(e);
                throw ae;
            }

            if (pVC.getResult()) {
                return username;
            }
        }
```

```
        return null;
    }

    private AuthStatus sendAuthenticateChallenge(MessageInfo msgInfo) {

        var realm = realmName;

        // if the realm property is set use it, otherwise use the name of the
server as the realm name.
        if (realm == null) {
            var request = (HttpServletRequest) msgInfo.getRequestMessage();
            realm = request.getServerName();
        }

        var response = (HttpServletResponse) msgInfo.getResponseMessage();

        response.setHeader(AUTHENTICATION_HEADER, BASIC + " realm=\"" + realm +
"\"");
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        return AuthStatus.SEND_CONTINUE;
    }

    public AuthStatus secureResponse(MessageInfo msgInfo, Subject service) throws
AuthException {
        return AuthStatus.SEND_SUCCESS;
    }

    public void cleanSubject(MessageInfo msgInfo, Subject subject) throws
AuthException {
        if (subject != null) {
            subject.getPrincipals().clear();
        }
    }

    // distinguish the caller principal and assign default groups
    private void setAuthenticationResult(String name, Subject s, MessageInfo m)
throws IOException, UnsupportedCallbackException {
        handler.handle(new Callback[]{
           new CallerPrincipalCallback(s, name)
        });
        if (name != null) {
           // add the default group if the property is set
           if (defaultGroup != null) {
               handler.handle(new Callback[]{
                   new GroupPrincipalCallback(s, defaultGroup)
               });
           }
           m.getMap().put(AUTH_TYPE_INFO_KEY, "MySAM");
        }
    }
}
```

NOTE	The `initialize` method looks for the `group.name` and `realm.name` properties. The `group.name` property configures the default group assigned as a result of any successful authentication.

> The `realm.name` property defines the realm value sent back to the browser in the `WWW-Authenticate` challenge.

Compiling and Installing a Server Authentication Module

Before you can use the sample SAM, you need to compile, install, and configure it. Then you can bind it to an application.

To compile the SAM, include the SPI in your classpath. After you compile the SAM, install it by copying a JAR file containing the compiled SAM to the `$as-install/glassfish/lib` directory.

Configuring a Server Authentication Module

You can configure a SAM in one of these ways:

- In the Administration Console, open the *Security* component under the relevant configuration and go to the *Message Security* page. Set the following options:
  - **Authentication Layer**: `HttpServlet`
  - **Provider Type**: `server` or `client-server`
  - **Provider ID**: Specify a unique name for the SAM, for example `MySAM`
  - **Class Name**: Specify the fully qualified class name, for example `fish.payara.samples.security.MySam`
  - **Additional Property**:
    - Name: `group-name`
    - Value: `user`
  - **Additional Property**:
    - Name: `realm-name`
    - Value: `Sam`
- Use the `asadmin create-message-security-provider` command to configure a SAM. Set the following options:

- `--layer HttpServlet`
- `--providertype server` or `--providertype client-server`
- `--classname fish.payara.samples.security.MySam`
- `--property group-name=user:realm-name=Sam`
- Provider name operand — Specify a unique name for the SAM, for example `MySAM`

Binding a Server Authentication Module to Your Application

After you install and configure the SAM, you can bind it for use by the container on behalf of one or more of your applications. You have two options in how you bind the SAM, depending on whether you are willing to repackage and redeploy your application:

- If you are willing to repackage and redeploy, you can bind the SAM using the `glassfish-web.xml` or `payara-web.xml` deployment descriptor file. Set the value of the `httpservlet-security-provider` attribute of the `glassfish-web-app`/`payara-web-app` element to the SAM's configured provider ID, for example, `MySAM`. This option leverages the native `AuthConfigProvider` implementation that ships with Payara Server.
- Another approach is to develop your own `AuthConfigProvider` and register it with the server's `AuthConfigFactory` for use on behalf of your applications. For example, a simple `AuthConfigProvider` can obtain, through its initialization properties, the classname of a SAM to configure on behalf of the applications for which the provider is registered.

TIP You can find a description of the functionality of an `AuthConfigProvider` and of the registration facilities provided by an `AuthConfigFactory` in the Jakarta Authentication specification.

IMPORTANT glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

Advanced Groups Configuration

Payara Server allows the advanced configuration mapping of a client certificate's distinguished name (DN) to specific realm groups, which allows the realm to automatically assign groups based on the values of each DN part to all principals registered in the realm.

This feature can be configured by using the `dn-parts-used-for-groups` property of any security realm of type `com.sun.enterprise.security.auth.realm.certificate.CertificateRealm`. It must define a comma separated list of identifiers of DN parts, i.e. `EMAILADDRESS,DC,OU`, and it is empty by default.

NOTE The default empty value means that only content of the `assign-groups` property value will be used as the list of assigned groups.

If both properties are set, a principal who completed the certificate validation will have all groups from both `assign-groups` settings.

Not all DN parts are supported. The following is the list of all parts that are supported, categorized by their OIDs:

Table 184. Supported OIDs

DN Part	Description	Object ID
CN	Common Name	2.5.4.3
SURNAME	Surname	2.5.4.4
SERIALNUMBER	Serial Number of the certificate	2.5.4.5
C	Country	2.5.4.6
L	Locality	2.5.4.7
ST	State	2.5.4.8
STREET	Street	2.5.4.9

DN Part	Description	Object ID
O	Organisation	2.5.4.10
OU	Organisation Unit	2.5.4.11
T	Title	2.5.4.12
GIVENNAME	Given Name	2.5.4.42
INITIALS	Initials	2.5.4.43
GENERATION	Generation	2.5.4.44
DNQUALIFIER	DN Qualifier	2.5.4.46
UID	User ID	0.9.2342.19200300.100.1.1
DC	Domain Component	0.9.2342.19200300.100.1.25
EMAILADDRESS	E-Mail address	1.2.840.113549.1.9.1
IP	IP Address	1.3.6.1.4.1.42.2.11.2.1

Configuration Using the Admin Console

This feature can be configured on the default certificate realm with the following steps:

1. Navigate to the applicable configuration page for your use (e.g. `server-config`) under the `Configurations` option in the side menu
2. Head to `Security` → `Realms` and select the `certificate` realm
3. Click the `Add Property` button
4. Set the property *Name* to be `dn-parts-used-for-groups` and set the *Value* to a comma separated list of OID names.

Configuration Using the Asadmin CLI

You can also use the following asadmin command to set the value of the property in the security realm:

```
asadmin set configs.config.${YOUR_INSTANCE_CONFIG}.security-service.auth-
realm.certificate.property.dn-parts-used-for-groups=EMAILADDRESS,DC,OU
```

IMPORTANT	After setting the value of the property, make sure that you **restart the server instance** for the changes to take effect.

CAUTION	Keep in mind that the set of DN parts used to configure the realm groups must be predictable to be useful, so be careful to use DN parts that do not create group name collisions.

Mapping Groups to Security Roles in Applications

After configuring the realm groups, you can map these groups to application roles as you'd do by using the Payara Server specific deployment descriptors. You can even map these roles to principals that are configured as documented in the advanced Principal Name configuration section.

Here's a quick summary of the authentication and authorization process would work in this scenario:

- The principal receives principal name as configured (whole DN or only CN part)
- The principal receives all groups from realm's `assign-groups` list
- The principal receives all groups from certificate's DN parts listed in `dn-parts-used-for-groups` realm property (limited to those listed in table)
- The server allows the principal to access the application
- The client receives all roles from application's security role mapping, where at least one of his principal names or groups matches their respective element.
- The security roles are then used for authorization to access application resources

For example, if user roles were mapped like this in the `payara-web.xml` deployment descriptor of an application, and we enabled using `CN` as a principal name and set the `dn-parts-used-for-groups` to `OU`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-web-app PUBLIC "-//Payara.fish//DTD Payara Server 7 Servlet 6.1
Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-web-app_6_1-1.dtd">
```

```
<payara-web-app error-url="">
  <context-root>/health-services</context-root>
  <security-role-mapping>
    <role-name>role1</role-name>
    <principal-name>C=UK, S=lak, L=zak, OU=unitA, CN=foo1</principal-name>
    <principal-name>C=UK, S=lak, L=zak, OU=unitA, CN=foo2</principal-name>
    <principal-name>C=UK, S=lak, L=zak, OU=unitC, CN=foo4</principal-name>
    <principal-name>C=UK, S=lak, L=zak, OU=unitD, CN=foo-director</principal-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>role2</role-name>
    <principal-name>C=UK, S=lak, L=zak, OU=unitB, CN=foo3</principal-name>
    <principal-name>C=UK, S=lak, L=zak, OU=unitD, CN=foo-director</principal-name>
  </security-role-mapping>
</payara-web-app>
```

Then the role mapping can be based on organizational unit in this case. But you can still use the principal name as in any normal case:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-web-app PUBLIC "-//Payara.fish//DTD Payara Server 7 Servlet 6.1
Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-web-app_6_1-1.dtd">
<payara-web-app error-url="">
  <context-root>/health-services</context-root>
  <security-role-mapping>
    <role-name>role1</role-name>
    <group-name>unitA</group-name>
    <group-name>unitC</group-name>
    <group-name>unitD</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>role2</role-name>
    <group-name>unitB</group-name>
    <principal-name>foo-director</principal-name>
  </security-role-mapping>
</payara-web-app>
```

Advanced Principal Name Configuration

Payara Server allows the use of the Common-Name (`CN`) attribute of a client certificate's DN to be used as the full principal's name when the certificated is authenticated in an application. This simplifies the authentication process, as the common name will replace the full name in the validation process of the certificate's authentication.

To configure this behaviour, the `common-name-as-principal-name` property can be used in any security realm of type `com.sun.enterprise.security.auth.realm.certificate.CertificateRea`

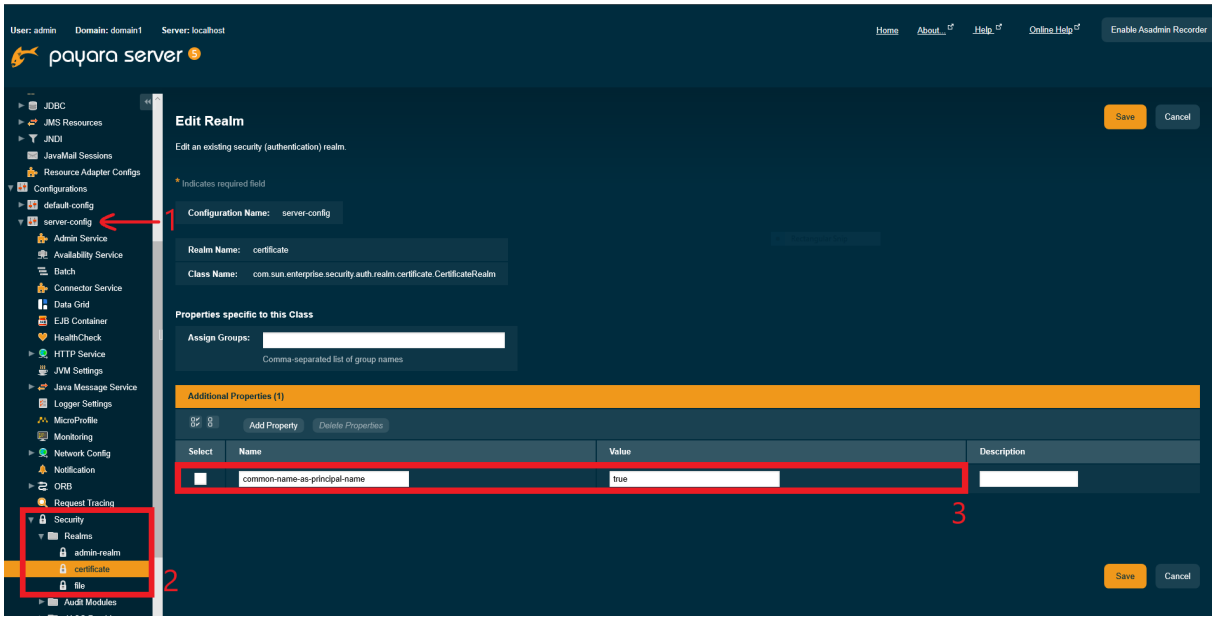
`lm`.

> **NOTE**
>
> The `common-name-as-principal-name` property is set to false by default if no configuration is provided.

Configuration Using the Admin Console

This feature can be configured on the default certificate realm as follows on the Admin Console:

1. Navigate to the applicable configuration page for your use (e.g. `server-config`) under the `Configurations` option in the side menu
2. Head to `Security` → `Realms` and select the `certificate` realm
3. Click the `Add Property` button
4. Set the property *Name* to be `common-name-as-principal-name` and set the *Value* to `true`



Configuration Using the Asadmin CLI

You can also use the following asadmin command to set the value of the property:

```
asadmin set configs.config.${YOUR_INSTANCE_CONFIG}.security-service.auth-
realm.certificate.property.common-name-as-principal-name=true
```

IMPORTANT	After setting the value of the property, make sure that you **restart the server instance** for the changes to take effect.

Using `CN` as Subject in Applications

Once this feature is enabled, users can be authenticated when a valid certificate uses the common name instead of the full domain name. It is also possible to map users to roles in this directly.

For example, if user authorization roles were mapped like this in the `payara-web.xml` deployment descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-web-app PUBLIC "-//Payara.fish//DTD Payara Server 7 Servlet 6.1
Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-web-app_6_1-1.dtd">
<payara-web-app error-url="">
  <context-root>/health-services</context-root>
  <security-role-mapping>
    <role-name>gl</role-name>
    <principal-name>C=UK, S=lak, L=zak, O=kaz, CN=foo</principal-name>
  </security-role-mapping>
</payara-web-app>
```

Then the `principal-name` can be simplified by using only the common name like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-web-app PUBLIC "-//Payara.fish//DTD Payara Server 7 Servlet 6.1
Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-web-app_6_1-1.dtd">
<payara-web-app error-url="">
  <context-root>/health-services</context-root>
  <security-role-mapping>
    <role-name>gl</role-name>
    <principal-name>foo</principal-name>
  </security-role-mapping>
</payara-web-app>
```

Client Certificate Custom Validations

By default, the Payara Server runtime checks only if the submitted Client Certificate is present in the trust store and that it hasn't expired already. If these conditions are met, it is accepted without any additional checks.

Custom Client Certificate Validation allows you to perform extra checks in addition to checking for the presence of the certificate in the trust store.

You can perform a check on the validity date, and use the Online Certificate Status Protocol (OCSP) to validate the certificate when it has a Certificate Revocation List entry (CRL), or lookup the DN name in a database for example, and determine based on this information if the certificate is still accepted.

Creating a Custom Validator

When you want to implement custom validation, implement the `fish.payara.security.client.ClientCertificateValidator` interface and define your class through the `ServiceLoader` mechanism.

To have the `ClientCertificateValidator` interface available in your application, add the `payara-api` artifact to your application with the provided scope. When using maven, add the following snippet if not already defined.

```
<dependency>
    <groupId>fish.payara.api</groupId>
    <artifactId>payara-api</artifactId>
    <version>{release-id}</version>
    <scope>provided</scope>
</dependency>
```

Now that you have the interface available, you can implement it in your application.

```
public class MyCertificateValidator implements ClientCertificateValidator {

    @Override
    public boolean isValid(Subject subject, X500Principal principal,
X509Certificate certificate) {
        var isValid = true;
        //Determine if the subject is valid
        return isValid;
    }
}
```

The most important parameter here is the `principal` parameter which contains the user information contained in the Client Certificate presented in the request. The Certificate itself is in the last parameter and in the case you like to have access already

to the `Subject` for this validation, it is passed in as the first parameter.

When you return `true` as the method result, the processing of the request continues. In the case false is returned, a `LoginException` is thrown resulting in a status `401` for the request.

This class is loaded through the Java ServiceLoader mechanism. Make sure you have the following file. `META-INF/services/fish.payara.security.client.ClientCertificateValidator` containing the fully qualified name of your implementation.

```
com.company.certificate.MyCertificateValidator
```

NOTE

You can @Inject CDI and EJB beans in your validator implementation, however since the validator itself is not a Jakarta EE bean, `@PostConstruct` and `@PreDestroy`-annotated methods will not be called on the validator instance.

Certificate Expiration Validation

By default, certificate expiration validation is enabled. This validation checks if the certificates in the trust store haven't expired yet. If the client certificate has expired or is not valid yet, a warning message will be printed in the server logs.

Configuration Using the Asadmin CLI

You can use the following asadmin command to set the value of the property through:

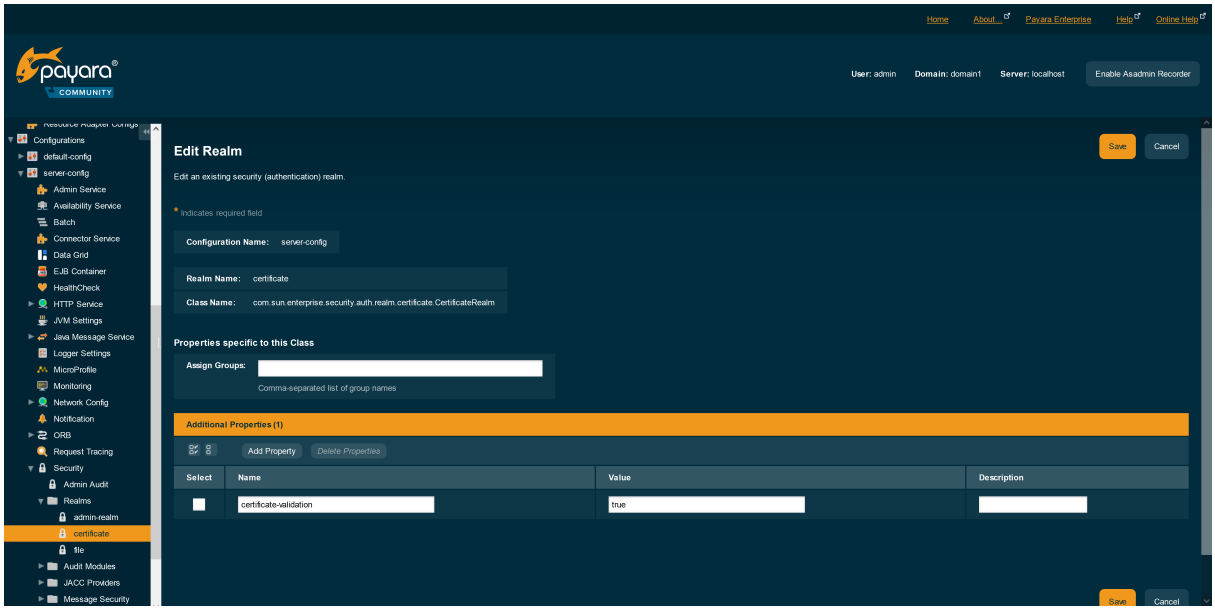
```
asadmin> set configs.config.${YOUR_INSTANCE_CONFIG}.security-service.auth-
realm.certificate.property.certificate-validation=[true/false]
```

Configuration Using the Admin Console

This feature can be configured on the default certificate realm as follows on the Admin Console:

1. Navigate to the applicable configuration page for your use (e.g. `server-config`) under the `Configurations` option in the side menu

2. Head to `Security` → `Realms` and select the `certificate` realm
3. Set the property `certificate-validation` *value* to `[true/false]`



IMPORTANT	After setting the value of the property, make sure that you **restart the server instance** for the changes to take effect.

JCE Provider Support

Payara Server supports using the JCE framework to configure custom security providers. Custom providers can be installed statically in the JDK, via the service loader mechanism, or via the programmatic API.

NOTE	Although the provider can be changed while the server is running, this can influence other processes that already use the previous preferred provider.

Example

The example below uses an existing provider (Bouncy Castle), to which a custom service is added programmatically. As suggested above, this is not the suggested way to do this since it is being changed dynamically. Rather, this example is here for an idea of how it *could* be used.

Custom Certificate Factory

```
public class MyJCECertificateFactory extends CertificateFactory {

    @Override
    public Certificate engineGenerateCertificate(InputStream in) throws
CertificateException {
        Certificate certificate = super.engineGenerateCertificate(in);

        if (certificate instanceof X509Certificate == false) {
            return certificate;
        }

        return new MyJCEX509Certificate((X509Certificate) certificate);
    }

}
```

This factory is based on the CertificateFactory SPI type, but instead of implementing it fully the Bouncy Castle's CertificateFactory is used.

The returned Certificate is then wrapped in our own. A custom certificate type might then look as follows:

Custom Certificate Type

```
public class MyJCEX509Certificate extends X509Certificate {

    private final X509Certificate certificate;

    public MyJCEX509Certificate(X509Certificate certificate) {
        this.certificate = certificate;
    }

    @Override
    public X500Principal getSubjectX500Principal() {

        var principal = certificate.getSubjectX500Principal();
        if ("C=UK,ST=lak,L=zak,O=kaz,OU=bar,CN=lfoo".equals(principal.getName())) {
            return new X500Principal("CN=u1");
        }
        return principal;
    }

    @Override
    public Principal getSubjectDN() {

        va principal = certificate.getSubjectDN();
        if ("CN=lfoo,OU=bar,O=kaz,L=zak,ST=lak,C=UK".equals(principal.getName())) {
            return new X500Principal("CN=u1");
        }
        return principal;
    }
```

```
    // Other methods omitted for brevity
}
```

This custom certificate is where the "principal mapping" is performed. For the example only `CN=lfoo,OU=bar,O=kaz,L=zak,ST=lak,C=UK` to `CN=u1` is mapped though.

> **NOTE**
>
> The example above overrides two methods. The `getSubjectDN()` method is actually deprecated (or denigrated as in the Certificate's terminology), but it's still being used. `getSubjectX500Principal()` is the preferred method to use.

Finally, the factory can be installed for example via a Servlet that we can easily ping (as mentioned, this would not normally be the advisory way to do this):

Custom Factory Installation

```
@WebServlet(urlPatterns = { "/BouncyServlet" })
public class BouncyServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

        var provider = new BouncyCastleProvider();
        provider.put("CertificateFactory.X.509", MyJCECertificateFactory.class
.getName());

        int pos = Security.insertProviderAt(provider, 1);
        response.getWriter().print("pos:" + pos);
    }
}
```

The above code instantiates the bouncy castle provider, and registers the previous custom certificate factory with it. In this example you can see the type and algorithm being combined into the single key for registering our factory class.

The modified provider is then inserted at provider position `1` (the first, there's no 0), using a static method of the Security class.

After pinging this Servlet, and requesting another Servlet over HTTPS that's protected with the client-cert authentication mechanism, a principal with name

`CN=lfoo,OU=bar,O=kaz,L=zak,ST=lak,C=UK` will be replaced at nearly the lowest level in the system with `CN=u1`, and the server runtime will only see this principal name. We can use this principal name for instance to map groups and roles to it:

```
<payara-web-app>
    <security-role-mapping>
        <role-name>g1</role-name>
        <group-name>g1</group-name>
        <principal-name>CN=u1</principal-name>
    </security-role-mapping>
</payara-web-app>
```

A fully working sample demonstrating exactly this can be found in the Java EE 7 samples repo.

Developing CDI Components

This section lists additional available enhancements to the CDI implementation shipped in Azul Payara.

Concurrent CDI Bean Loading

It is possible to concurrently load CDI beans and potentially speed up an application's load time by enabling Weld's Multi-Threaded Bean Loading feature.

IMPORTANT	By default, this feature is disabled for all applications on Payara Server, and enabled for all applications on Payara Micro.

Enable Concurrent CDI Bean Loading

Concurrent CDI bean loading is configured on a per-configuration basis using the `set` on the `cdi-service.enable-concurrent-deployment` property like this:

```
asadmin set configs.config.server-config.cdi-service.enable-concurrent-deployment
=true
```

It is possible to configure the number of pre-loader threads like this:

```
asadmin set configs.config.server-config.cdi-service.pre-loader-thread-pool-size=4
```

When configuring the number of the pre-loader threads, use a sensible value that is consistent with the number of CPU cores available for the server instance, otherwise this setting may introduce performance degradation on the deployment times.

NOTE	Concurrent deployment of CDI Beans is enabled by default with 2 pre-loader threads in Payara Micro.

WARNING	There are no equivalent configuration views for these commands in the Web admin console. They'll be reserved for a future release.

Legacy Mode of Empty beans.xml

Starting with CDI 4.0, bean archives with empty beans.xml have discovery mode `annotated`. It is possible to activate the legacy mode option to switch this back to `all` discovery mode.

Enable Legacy Mode of Empty beans.xml

The legacy mode of empty beans.xml can be enabled using the following mechanisms:

Via Asadmin CLI

Using the asadmin CLI it is possible to enable legacy mode for a single application or for ALL deployed applications.

Enabling for a Single Application

The legacy mode can be enabled for a single application with the `deploy` command. Add the `emptyBeansXmlModeALL=true` property to the `--properties` argument's value of the `deploy` command.

Here's an example of enabling development mode for an application named `application.war`:

```
asadmin deploy --properties emptyBeansXmlModeALL=true myapplication.war
```

Legacy mode is configured on a per-configuration basis using the `create-system-properties` on the `fish.payara.deployment.emptyBeansXmlModeALL`

property like this:

```
asadmin create-system-properties fish.payara.deployment.emptyBeansXmlModeALL=true
```

NOTE	The legacy mode is enabled by default for any transformed Jakarta EE 8 applications that are converted via the Eclipse Transformer.

Developing SOAP Web Services

This chapter describes Payara Server's support for SOAP-compatible web services. "Building with Jakarta XML Web Services" in The Jakarta EE Tutorial shows more details on creating dedicated SOAP Web Services for any Jakarta EE runtime.

For additional information about JAXB (Jakarta XML Binding), see its corresponding specification and The Databinding Provider.

For additional information about Jakarta XML Web Services, see its corresponding specification.

For information about web services security, see configuring-message-security-for-web-services.

The Fast Infoset standard specifies a binary format based on the XML Information Set. This format is an efficient alternative to XML. For more information about using Fast Infoset, see the following links:

- Fast Infoset Project
- XML Information Set

Deploying a Web Service

You deploy a web service endpoint to Payara Server just as you would any servlet, stateless session bean (SLSB), or application.

IMPORTANT	For complex services with dependent classes, user specified WSDL files, or other advanced features, auto-deployment of an

annotated file is not sufficient.

The Payara Server deployment descriptor files `glassfish-web.xml`/`payara-web.xml` and `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` provide optional web service enhancements in the `webservice-endpoint` and `webservice-description` elements, including a `debugging-enabled` sub-element that enables the creation of a test page.

IMPORTANT glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

The test page feature is enabled by default and described in The Web Service URI, WSDL File, and Test Page.

The Web Service URI, WSDL File, and Test Page

Clients can run a deployed web service by accessing its service endpoint address URI, which has the following format:

```
http://${host}:${port}/${context-root}/${servlet-mapping-url-pattern}
```

The context-root is defined in the `application.xml` or `web.xml` file, and can be overridden in the `glassfish-application.xml`/`payara-application.xml` or `glassfish-web.xml`/`payara-web.xml` file. The servlet-mapping-url-pattern is defined in the `web.xml` file.

IMPORTANT glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

In the following example, the context-root is `my-ws` and the servlet-mapping-url-pattern is `/simple` for local deployment in the default port:

```
http://localhost:8080/my-ws/simple
```

You can view the WSDL file of the deployed service in a browser by adding `?WSDL` to the

end of the URI. For example:

```
http://localhost:8080/my-ws/simple?WSDL
```

For debugging, you can run a test page for the deployed service in a browser by adding `?Tester` to the end of the URL. For example:

```
http://localhost:8080/my-ws/simple?Tester
```

This URL will show the SOAP Web Service Tester in the corresponding browser allowing users to quickly test their web services after deployment.

You can also test a service using the Administration Console. Open the *Web Services* component for the corresponding deployed application, select the web service in the listing on the *General* tab, and select *Test*.

WARNING	The tester webpage works only for WS-I compliant web services. This means that the tester servlet does not work for services with WSDL files that use RPC/encoded binding.

The Web Service tester application is enabled by default. You can disable the test page for a web service by setting the value of the `debugging-enabled` element in the `glassfish-web.xml`/`payara-web.xml` and `glassfish-ejb-jar.xml`/payara-ejb-jar.xml` deployment descriptor to `false`.

The Databinding Provider

To specify the Jakarta XML Binding provider implementation for the server runtime, set the `com.sun.xml.ws.spi.db.BindingContextFactory` JVM property to one of the following values:

- `com.sun.xml.ws.db.glassfish.JAXBRIContextFactory`: Specifies the JAXB reference implementation in Payara Server (bundled within Metro). This is the default setting.

- `org.eclipse.persistence.jaxb.JAXBContextFactory`: Specifies Eclipselink MOXy Jakarta XML bindings.

  For example:

```
asadmin create-jvm-options -Dcom.sun.xml.ws.spi.db.BindingContextFactory
=org.eclipse.persistence.jaxb.JAXBContextFactory
```

To specify the Jakarta XML Binding provider for a **specific** web service endpoint, you have the following options:

- Set the `com.oracle.webservices.api.databinding.DatabindingModeFeature` feature during `WebServiceFeature` initialization or using the `add` method.

  Allowed values are as follows:

  - `com.oracle.webservices.api.databinding.DatabindingModeFeature.GLASSFISH_JAXB`: Specifies the JAXB reference implementation (bundled within Metro). This is the default setting.
  - `org.eclipse.persistence.jaxb.JAXBContextFactory.ECLIPSELINK_JAXB`: Specifies Eclipselink MOXy JAXB binding.

    For example:

```
import jakarta.xml.ws.WebServiceFeature;
import com.oracle.webservices.api.databinding.DatabindingModeFeature;
import org.eclipse.persistence.jaxb.JAXBContextFactory;

WebServiceFeature[] features = {new DatabindingModeFeature(
JAXBContextFactory.ECLIPSELINK_JAXB)};
```

- Set the `com.oracle.webservices.api.databinding.DatabindingModeFeature` feature using the `@DatabindingMode` annotation. For example:

```
import jakarta.jws.WebService;
import com.oracle.webservices.api.databinding.DatabindingMode;
```

```
import org.eclipse.persistence.jaxb.JAXBContextFactory;

@WebService
@DatabindingMode(JAXBContextFactory.ECLIPSELINK_JAXB)
public class MyWebService{
}
```

IMPORTANT	To configure these features in your application at compile time you'll have to configure explicit dependencies to the Eclipse EE4J Metro and EclipseLink's projects.

Additional Scanning Locations

Payara Server scans for specific JAX-WS files in locations further to those specified in the **JAX-WS** specification.

Additional directories and sources are scanned are documented here.

`jax-ws-catalog.xml`

The Jakarta XML Web Services specification states that the `jax-ws-catalog.xml` file should be searched for in the `META-INF` directory of a deployed `WAR` file relative to the classpath.

Since the classpath of a WAR can be found in `/WEB-INF/classes/`, this means that this file is expected at `/WEB-INF/classes/META-INF/jax-ws-catalog.xml` (from the root of the `WAR` artifact).

NOTE	Payara Server also searches for this file at `/WEB-INF/jax-ws-catalog.xml`.

Configuring the Java Persistence Provider

This chapter describes EclipseLink, the default persistence provider in Payara Server, and introduces how to use it.

This chapter also tells how to set the default persistence provider in Payara Server and how to use persistence-related features specific to Azul Payara such as automatic schema generation.

Eclipselink Overview

EclipseLink is the default persistence provider in Azul Payara. It is a comprehensive standards-based object-persistence and object-transformation framework that provides APIs, schemas, and run-time services for the persistence layer of an application.

EclipseLink implements the following specifications, plus value-added extensions:

Jakarta Persistence (JPA) 3.0+:

JPA is part of the Jakarta EE specification. It includes improvements and enhancements to domain modeling, object/relational mapping, `EntityManager` and `Query` interfaces, and the Jakarta Persistence Query Language (JPQL). It includes an API for criteria queries, a metamodel API, and support for validation. The Jakarta Persistence API can be used with non-EJB components outside the EJB container.

+ For more information, see the Jakarta Persistence specification. For basic information about the Jakarta Persistence API, see "Persistence" in The Jakarta EE Tutorial.

Jakarta XML Binding (JAXB): The EclipseLink JAXB implementation, plus EclipseLink extensions, is called **MOXy**. The `org.eclipse.persistence.moxy.jar` file is bundled as a module in Payara Server.

+ For more information about MOXy support in Payara Server, see the-databinding-provider.

+ For the JAXB specification, see the Jakarta XML Binding specification.

- EclipseLink binary utilities are not included but can be used with Payara Server. To use them, download the EclipseLink zip file at `http://www.eclipse.org/eclipselink/downloads/` and unzip it. The utility files are located in the `bin` folder and are the following:
  - `jaxb-compiler`: Used to compile Java binding source files.
  - `sdo-compiler`: Used to compile SDO files.
  - `sdo-jaxb-compiler`: Used to compile Java binding source files for SDO

components.

NOTE	All utilities are available as both a CMD and BASH shell script.

For information about developing, deploying, and configuring EclipseLink, and applications, see the following resources:

- EclipseLink Project Home
- EclipseLink Documentation Center
- EclipseLink JPA Examples

Using Eclipselink in Payara Server

To run EclipseLink JPA applications in Payara Server, you must configure the server and coordinate certain server and application settings. These are described in the following steps:

1. Set up the datasource. See "administering-database-connectivity" in Payara Server General Administration section.
2. Create the application. For guidance in writing your application, see "Persistence" in The Jakarta EE Tutorial.
3. Create the `persistence.xml` file. See Specifying the Database for an Application for considerations specific to Payara Server.

   If you are using the Jakarta Persistence API by calling `Persistence.createEMF()`, see Specifying the Persistence Provider for an Application.

4. If the security manager is enabled, and you are using the Jakarta Persistence API by calling `Persistence.createEMF()`, see enabling-and-disabling-the-security-manager.
5. Deploy the application.
6. Run the application.

7. Monitor the application. See "administering-the-monitoring-service" in the Payara Server General Administration section.

Specifying the Database for an Application

Payara Server uses an embedded H2 database to provision its default datasource, named `jdbc/__default`. If the `transaction-type` element is omitted or specified as `JTA` and both the `jta-data-source` and `non-jta-data-source` elements are omitted in the `persistence.xml` file, the default datasource is used as a JTA data source.

If `transaction-type` is specified as `RESOURCE_LOCAL` and both `jta-data-source` and `non-jta-data-source` are omitted, the default datasource is used as a non-JTA data source.

TIP

To use a non-default database, either specify a value for the `jta-data-source` element, or set the `transaction-type` element to `RESOURCE_LOCAL` and specify a value for the `non-jta-data-source` element.

When using EclipseLink, it attempts to automatically detect the database type based on the connection metadata. This database type is used to issue SQL statements specific to the detected database type's SQL dialect.

You can specify the optional `eclipselink.target-database` property to guarantee that the database type is correct. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="3.0" xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd">
    <persistence-unit name ="em1">
        <jta-data-source>jdbc/MyDB2DB</jta-data-source>
        <properties>
            <property name="eclipselink.target-database"  value="DB2"/>
        </properties>
    </persistence-unit>
</persistence>
```

The following `eclipselink.target-database` property values are allowed. Supported platforms have been tested in Payara Server and are found to be Jakarta EE compatible.

```
//Supported platforms
Attunity
Cloudscape
DB2
DB2Mainframe
DBase
Derby
HSQL
H2
Informix
JavaDB
MaxDB
MySQL
MySQL4
Oracle
Oracle10
Oracle11
Oracle8
Oracle9
PointBase
PostgreSQL
SQLAnywhere
SQLServer
Sybase
Symfoware
TimesTen
```

For more information about the `eclipselink.target-database` property, see EclipseLink JPA Extensions - Target Database.

If you are using the Java Persistence API by calling `Persistence.createEMF()`, do not specify the `jta-data-source` or `non-jta-data-source` elements. Instead, specify the `provider` element and any additional properties required by the JDBC driver or the database. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="3.0" xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd">
    <persistence-unit name ="em2">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <class>fish.payara.jpa.JpaBean</class>
        <properties>
            <property name="eclipselink.target-database" value="H2"/>
```

```
            <!-- JDBC connection properties -->
            <property name="eclipselink.jdbc.driver" value="org.h2.Driver"/>
            <property name="eclipselink.jdbc.url"
value="jdbc:h2://localhost:1527/~/testdb"/>
            <property name="eclipselink.jdbc.user" value="APP"/>
            <property name="eclipselink.jdbc.password" value="APP"/>
        </properties>
    </persistence-unit>
</persistence>
```

For configurations of supported and other drivers, see "configuration-specifics-for-jdbc-drivers" in the Payara Server General Administration section.

Specifying the Persistence Provider for an Application

If you are using the default persistence provider in an application that uses the Jakarta Persistence API by injecting or looking up an entity manager or entity manager factory, you do not need to specify the provider explicitly.

If you are using the Jakarta Persistence API by calling `Persistence.createEMF()`, you should always specify the persistence provider for specification compliance. To specify the default provider, set the `provider` element of the `persistence.xml` file to `org.eclipse.persistence.jpa.PersistenceProvider`.

You can specify a non-default persistence provider for an application in the manner described in the Jakarta Persistence API Specification:

1. Install the provider. Copy the provider JAR files to the `domain-dir/lib` directory, and restart the Payara Server domain.

   The new persistence provider is now available to all modules and applications deployed on servers that share the same configuration.

   NOTE | EclipseLink will remain as the default persistence provider.

2. In your persistence unit, specify the provider and any properties the provider requires in the `persistence.xml` file. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence">
```

```
    <persistence-unit name ="em3">
        <provider>com.company.persistence.PersistenceProviderImpl</provider>
        <properties>
            <property name="company.database.name" value="MyDB"/>
        </properties>
    </persistence-unit>
</persistence>
```

Primary Key Generation Defaults

In the descriptions of the `@GeneratedValue`, `@SequenceGenerator`, and `@TableGenerator` annotations in the Jakarta Persistence Specification, certain defaults are noted as specific to the persistence provider. The default persistence provider's primary key generation defaults are listed here.

`@GeneratedValue` defaults are as follows:

- Using `strategy=AUTO` (or no `strategy`) creates a `@TableGenerator` named `SEQ_GEN` with default settings. Specifying a `generator` has no effect.
- Using `strategy=TABLE` without specifying a `generator` creates a `@TableGenerator` named `SEQ_GEN_TABLE` with default settings.

  Specifying a `generator` but no `@TableGenerator` creates and names a `@TableGenerator` with default settings.

- Using `strategy=IDENTITY` or `strategy=SEQUENCE` produces the same results, which are database-specific.
  - For Oracle databases, not specifying a `generator` creates a `@SequenceGenerator` named `SEQ_GEN_SEQUENCE` with default settings.

    Specifying a `generator` but no `@SequenceGenerator` creates and names a `@SequenceGenerator` with default settings.

  - For PostgreSQL databases, a `SERIAL` column named entity-table`_`pk-column`_SEQ` is created.
  - For MySQL databases, an `AUTO_INCREMENT` column is created.
  - For other supported databases, an `IDENTITY` column is created.

The `@SequenceGenerator` annotation has one default specific to the default provider. The default `sequenceName` is the specified `name`.

`@TableGenerator` defaults are as follows:

- The default `table` is `SEQUENCE`.
- The default `pkColumnName` is `SEQ_NAME`.
- The default `valueColumnName` is `SEQ_COUNT`.
- The default `pkColumnValue` is the specified `name`, or the default `name` if no `name` is specified.

Automatic Schema Generation

The automatic schema generation feature of Payara Server defines database tables based on the fields or properties in entities and the relationships between the fields or properties.

This insulates developers from many of the database related aspects of development, allowing them to focus on entity development.

The resulting schema is usable as-is or can be given to a database administrator for tuning with respect to performance, security, and so on.

IMPORTANT	Automatic schema generation is supported on an all-or-none basis: it expects that no tables exist in the database before it is executed. It is not intended to be used as a tool to generate extra tables or constraints. Deployment won't fail if all tables are not created, and un-deployment won't fail if not all tables are dropped. Instead, an error is written to the server log. This is done to allow you to investigate the problem and fix it manually. You should not rely on the partially created database schema to be correct for running the application.

Annotations

The following annotations are used in automatic schema generation:

- `@AssociationOverride`
- `@AssociationOverrides`
- `@AttributeOverride`
- `@AttributeOverrides`
- `@Column`
- `@DiscriminatorColumn`
- `@DiscriminatorValue`
- `@Embedded`
- `@EmbeddedId`
- `@GeneratedValue`
- `@Id`
- `@IdClass`
- `@JoinColumn`
- `@JoinColumns`
- `@JoinTable`
- `@Lob`
- `@ManyToMany`
- `@ManyToOne`
- `@OneToMany`
- `@OneToOne`
- `@PrimaryKeyJoinColumn`
- `@PrimaryKeyJoinColumns`

- `@SecondaryTable`
- `@SecondaryTables`
- `@SequenceGenerator`
- `@Table`
- `@TableGenerator`
- `@UniqueConstraint`
- `@Version`

For information about these annotations, see the Jakarta Persistence Specification.

NOTE	For `@Column` annotations, the `insertable` and `updatable` elements are not used in automatic schema generation.
WARNING	For `@OneToMany` and `@ManyToOne` annotations, no `ForeignKeyConstraint` is created in the resulting DDL files.

Generation Options

Schema generation properties or `asadmin` command line options can control automatic schema generation by the following:

- Creating tables during deployment
- Dropping tables during un-deployment
- Dropping and creating tables during redeployment
- Generating the DDL files

Optional schema generation properties control the automatic creation of database tables. You can specify them in the `persistence.xml` file. For more information, see EclipseLink JPA Extensions for Schema Generation.

The following options of the `asadmin deploy` command control the automatic creation of database tables at deployment.

Table 185. The `asadmin deploy` Generation Options

Option	Default	Description
`--createtables`	none	If `true`, causes database tables to be created for entities that need them. No unique constraints are created. If `false`, does not create tables. If not specified, the value of the `eclipselink.ddl-generation` property in `persistence.xml` is used.
`--dropandcreatetables`	none	If `true`, and if tables were automatically created when this application was last deployed, tables from the earlier deployment are dropped and fresh ones are created. If `true`, and if tables were not automatically created when this application was last deployed, no attempt is made to drop any tables. If tables with the same names as those that would have been automatically created are found, the deployment proceeds, but a warning is thrown to indicate that tables could not be created. If `false`, the `eclipselink.ddl-generation` property setting in `persistence.xml` is overridden.

The following options of the `asadmin undeploy` command control the automatic removal of database tables at un-deployment.

Table 186. The `asadmin undeploy` Generation Options

Option	Default	Description
`--droptables`	none	If `true`, causes database tables that were automatically created when the entities were last deployed to be dropped when the entities are un-deployed. If `false`, does not drop tables. If not specified, tables are dropped only if the `eclipselink.ddl-generation` property setting in `persistence.xml` is `drop-and-create-tables`.

CAUTION When `asadmin` deployment options and `persistence.xml` options are both specified, the `asadmin` deployment options take precedence.

Restrictions and Optimizations

This section discusses restrictions and performance optimizations that affect using the Java Persistence API.

Extended Persistence Context

The Jakarta Persistence API specification does not specify how the container and persistence provider should work together to serialize an extended persistence context. This also prevents successful serialization of a reference to an extended persistence context in a stateful session bean.

Even in a single-instance environment, if a stateful session bean is passivated, its extended persistence context could be lost when the stateful session bean is activated.

In Payara Server, a stateful session bean with an extended persistence context is never passivated and cannot be failed over.

Using `@OrderBy` with a Shared Session Cache

Setting `@OrderBy` on a `ManyToMany` or `OneToMany` relationship field in which a `List` represents the Many side doesn't work if the session cache is shared. Use one of the following workarounds:

- Have the application maintain the order so the `List` is cached properly.
- Refresh the session cache using `EntityManager.refresh()` if you don't want to maintain the order during creation or modification of the `List`.
- Disable session cache sharing in `persistence.xml` as follows:

```
<property name="eclipselink.cache.shared.default" value="false"/>
```

Database Case Sensitivity

Mapping references to column or table names must be in accordance with the expected column or table name case, and ensuring this is the programmer's responsibility.

If column or table names are not explicitly specified for a field or entity, Payara Server uses upper case column names by default, so any mapping references to the column or table names must be in upper case.

If column or table names are explicitly specified, the case of all mapping references to the column or table names must be in accordance with the case used in the specified names.

The following are examples of how case sensitivity affects mapping elements that refer to columns or tables. Keep case sensitivity in mind when writing these mappings.

Unique Constraints

If column names are not explicitly specified on a field, unique constraints and foreign key mappings must be specified using uppercase references. For example:

```
@Table(name="Department",
    uniqueConstraints= {
        @UniqueConstraint(columnNames= "DEPTNAME")
})
```

The other way to handle this is by specifying explicit column names for each field with the required case. For example:

```
@Table(name="Department",
        uniqueConstraints={ @UniqueConstraint ( columnNames= { "deptName" } ) } )
```

```
public class Department{

    @Column(name="deptName")
    private String deptName;
}
```

Otherwise, the `ALTER TABLE` statement generated by Payara Server uses the incorrect case, and the creation of the unique constraint fails.

Foreign Key Mapping

Use `@OneToMany(mappedBy="COMPANY")` or specify an explicit column name for the `Company` field on the `Many` side of the relationship.

SQL Result Set Mapping

Use the following elements:

```
<sql-result-set-mapping name="SRSMName">
   <entity-result entity-class="entities.someEntity" />
   <column-result name="UPPERCASECOLUMNNAME" />
</sql-result-set-mapping>
```

Or specify an explicit column name for the `upperCaseColumnName` field.

Named Native Queries and JDBC Queries

Column or table names specified in SQL queries must be in accordance with the expected case. For example, MySQL requires column names in the `SELECT` clause of JDBC queries to be uppercase, while PostgreSQL and Sybase require table names to be uppercase in all JDBC queries.

Postgresql Case Sensitivity

PostgreSQL stores column and table names in lower case. JDBC queries on PostgreSQL retrieve column or table names in lowercase unless the names are quoted. For example:

```
use aliases Select m.ID AS \"ID\" from Department m
```

Use the backslash as an escape character in the class file, but not in the `persistence.xml` file.

Sybase Finder Limitation

If a finder method with an input greater than 255 characters is executed and the primary key column is mapped to a `VARCHAR` column, Sybase attempts to convert type `VARCHAR` to type `TEXT` and generates the following error:

```
com.sybase.jdbc2.jdbc.SybSQLException: Implicit conversion from datatype 'TEXT' to
'VARCHAR' is not allowed. Use the CONVERT function to run this query.
```

To avoid this error, make sure the finder method input is less than 255 characters.

Mysql Database Restrictions

The following restrictions apply when you use a MySQL database with Payara Server in conjunction with JPA.

- MySQL treats `int1` and `int2` as reserved words. If you want to define `int1` and `int2` as fields in your tables, use `\`int1\`` and `\`int2\`` field names in your SQL file.
- When `VARCHAR` fields get truncated, a warning is displayed instead of an error. To get an error message, start the MySQL database in strict SQL mode.
- The order of fields in a foreign key index must match the order in the explicitly created index on the primary table.
- The `CREATE TABLE` syntax in the SQL file must end with the following line.

  ```
  )  Engine=InnoDB;
  ```

  `InnoDB` provides MySQL with a transaction-safe (ACID compliant) storage engine having commit, rollback, and crash recovery capabilities.

- For a `FLOAT` type field, the correct precision must be defined. By default, MySQL uses four bytes to store a `FLOAT` type that does not have an explicit precision definition. For example, this causes a number such as `12345.67890123` to be rounded off to `12345.7` during an `INSERT`.

  To prevent this, specify `FLOAT(10,2)` in the DDL file, which forces the database to

use an eight-byte double-precision column. For more information, see `https://dev.mysql.com/doc/refman/8.0/en/numeric-types.html`.

- To use `||` as the string concatenation symbol, start the MySQL server with the `--sql-mode="PIPES_AS_CONCAT"` option.
- MySQL always starts a new connection when `autoCommit==true` is set. This ensures that each SQL statement forms a single transaction on its own. If you try to rollback or commit an SQL statement, you get an error message like the following:

```
javax.transaction.SystemException: java.sql.SQLException: Can't call rollback
when autocommit=true
javax.transaction.SystemException: java.sql.SQLException: Error open transaction
is not closed
```

  To resolve this issue, add `relaxAutoCommit=true` to the JDBC URL.

- MySQL does not allow a `DELETE` on a row that contains a reference to itself. Here is an example that illustrates the issue:

```
create table EMPLOYEE (
    empId   int          NOT NULL,
    salary  float(25,2) NULL,
    mgrId   int          NULL,
    PRIMARY KEY (empId),
    FOREIGN KEY (mgrId) REFERENCES EMPLOYEE (empId)
) ENGINE=InnoDB;

insert into Employee values (1, 1234.34, 1);
delete from Employee where empId = 1;
```

  This example fails with the following error message.

```
ERROR 1217 (23000): Cannot delete or update a parent row: a foreign key
constraint fails
```

  To resolve this issue, change the table creation script to the following:

```
create table EMPLOYEE (
    empId   int          NOT NULL,
    salary  float(25,2) NULL,
    mgrId   int          NULL,
    PRIMARY KEY (empId),
```

```
    FOREIGN KEY (mgrId) REFERENCES EMPLOYEE (empId)
    ON DELETE SET NULL
) ENGINE=InnoDB;

insert into Employee values (1, 1234.34, 1);
delete from Employee where empId = 1;
```

This can be done only if the foreign key field is allowed to be `null`.

Using the Hazelcast Eclipselink Cache Coordination Protocol

If necessary, your can configure the Data Grid (implemented via Hazelcast) as the coordination mechanism for the application's L2 cache (if enabled). To do so, configure the `persistence.xml` deployment descriptor as follows:

1. Set the `eclipselink.cache.coordination.protocol` property to `fish.payara.persistence.eclipselink.cache.coordination.HazelcastPublishingTransportManager`
2. Set the `eclipselink.cache.coordination.channel` property to a unique value for the coordination channel's name.

The channel property is optional, but is highly recommended if the application has more than one Persistence Unit. A good practice is to name the channel after the persistence unit.

Here is a complete example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence version="3.0" xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd">
    <persistence-unit name="MyPU" transaction-type="JTA">
        <jta-data-source>jdbc/myDataSource</jta-data-source>
        <shared-cache-mode>DISABLE_SELECTIVE</shared-cache-mode>
        <properties>
            <property name="eclipselink.cache.coordination.protocol"
value="fish.payara.persistence.eclipselink.cache.coordination.HazelcastPublishingTr
ansportManager"/>
            <property name="eclipselink.cache.coordination.channel"
value="MyPUChannel"/>
        </properties>
    </persistence-unit>
</persistence>
```

Developing Web Applications

This chapter describes how web applications are supported in Azul Payara.

For general information about web applications, see "The Web Tier" in The Jakarta EE Tutorial.

Using Servlets

Both Payara Server and Payara Micro supports the Jakarta Servlet specification. For information about using the Jakarta Servlet API, see the specification at https://jakarta.ee/specifications/servlet/.

This section describes how to create effective servlets to control application interactions running on Payara Server or Micro, including standard-based servlets. In addition, this section describes additional features introduced in Azul Payara.

Caching Servlet Results

Payara Server and Micro can cache the results of invoking a servlet, a JSP, or any URL pattern to make subsequent invocations of the same servlet, JSP, or URL pattern faster. The server runtime caches the request results for a specific amount of time. In this way, if another data call occurs, the server runtime can return the cached data instead of performing the operation again.

For example, if your servlet returns a stock quote that updates every **5 minutes**, you set the cache to expire after **300 seconds**.

Whether to cache results and how to cache them depends on the data involved. For example, it makes no sense to cache the results of a quiz submission, because the input to the servlet is different each time. However, it makes sense to cache a high level report showing demographic data taken from quiz results that is updated once an hour.

To define how a web application handles response caching, you'll have to modify specific attributes in the `glassfish-web.xml` or `payara-web.xml` deployment descriptor file.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.
WARNING	A servlet that uses caching is not portable.

For information about JSP caching, see JSP Caching.

Caching Features

Azul Payara has the following web application response caching capabilities:

- Caching is configurable based on the servlet name or the URI.
- When caching is based on the URI, this includes user specified parameters in the query string.

  For example, a response from `/garden/catalog?category=roses` is different from a response from `/garden/catalog?category=lilies`. These responses are stored under different keys in the cache.

- Cache size, entry timeout, and other caching behaviors are configurable.
- Entry timeout is measured from the time an entry is created or refreshed. To override this timeout for an individual cache mapping, specify the `cache-mapping` sub-element `timeout`.
- To determine caching criteria programmatically, write a class that implements the `com.sun.appserv.web.cache.CacheHelper` interface. For example, if only a servlet knows when a back end data source was last modified, you can write a helper class to retrieve the last modified timestamp from the data source and decide whether to cache the response based on that timestamp.
- To determine cache key generation programmatically, write a class that implements the `com.sun.appserv.web.cache.CacheKeyGenerator` interface. See The CacheKeyGenerator Interface.
- All non-ASCII request parameter values specified in cache key elements must be URL encoded. The caching subsystem attempts to match the raw parameter values in the

request query string.

- Since newly updated classes impact what gets cached, the web container clears the cache during dynamic deployment or reloading of classes.
- The following `HttpServletRequest` request attributes are exposed:
  - `com.sun.appserv.web.cachedServletName`, the cached servlet target
  - `com.sun.appserv.web.cachedURLPattern`, the URL pattern being cached
- Results produced by resources that are the target of a `RequestDispatcher.include()` or `RequestDispatcher.forward()` call are cached if caching has been enabled for those resources. For details, see "cache-mapping" and "dispatcher in the Payara Server Application Deployment section.

Default Cache Configuration

If you enable caching but do not provide any special configuration for a servlet or JSP page, the default cache configuration is as follows:

- The default cache timeout is **30 seconds**.
- Only the HTTP `GET` method is eligible for caching.
- HTTP requests with cookies or sessions automatically disable caching.
- No special consideration is given to `Pragma`, `Cache-control`, or `Vary` HTTP headers.
- The default key consists of the servlet's path (minus `pathInfo` and the query string).
- The "least recently used" list is maintained to evict cache entries if the maximum cache size is exceeded.
- Key generation concatenates the servlet path with key field values, if any are specified.
- Results produced by resources that are the target of a `RequestDispatcher.include()` or `RequestDispatcher.forward()` call are never cached.

Caching Example

Here is an example cache element in the `payara-web.xml` file:

```
<cache max-capacity="8192" timeout="60">
 <cache-helper name="myHelper" class-name="MyCacheHelper"/>
 <cache-mapping>
     <servlet-name>myservlet</servlet-name>
     <timeout name="timefield">120</timeout>
     <http-method>GET</http-method>
     <http-method>POST</http-method>
 </cache-mapping>
 <cache-mapping>
     <url-pattern> /catalog/* </url-pattern>
     <!-- cache the best-selling category; cache the responses to this resource
only when the given parameters exist.
         Cache only when the catalog parameter has 'lilies' or 'roses' but no other
catalog varieties:
         * /orchard/catalog?best&category='lilies'
         * /orchard/catalog?best&category='roses'

         but not the result of
         * /orchard/catalog?best&category='wild'
     -->
     <constraint-field name='best' scope='request.parameter'/>
     <constraint-field name='category' scope='request.parameter'>
         <value> roses </value>
         <value> lilies </value>
     </constraint-field>
      <!-- Specify that a particular field is of given range but the field doesn't
need to be present in all the requests -->
     <constraint-field name='SKUnum' scope='request.parameter'>
         <value match-expr='in-range'> 1000 - 2000 </value>
     </constraint-field>
     <!-- cache when the category matches with any value other than a specific
value -->
     <constraint-field name="category" scope="request.parameter">
         <value match-expr="equals" cache-on-match-failure="true">bogus</value>
     </constraint-field>
 </cache-mapping>
 <cache-mapping>
     <servlet-name> InfoServlet </servlet-name>
     <cache-helper-ref>myHelper</cache-helper-ref>
 </cache-mapping>
</cache>
```

For more information about the `glassfish-web.xml`/`payara-web.xml` caching settings, see "cache" in the Payara Server Application Deployment section.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

The Cachekeygenerator Interface

The built-in default `CacheHelper` implementation allows web applications to customize the key generation. An application component (in a servlet or JSP) can set up a custom `CacheKeyGenerator` implementation as an attribute in the `ServletContext`.

The name of the context attribute is configurable as the `value` of the `cacheKeyGeneratorAttrName` property in the `default-helper` element of the `glassfish-web.xml`/`payara-web.xml` deployment descriptor.

For more information, see "default-helper" in the Payara Server Application Deployment section.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

About the Servlet Engine

Servlets exist in and are managed by the servlet engine in Payara Server or Micro. The servlet engine is an internal container that handles all servlet meta functions. These functions include instantiation, initialization, destruction, access from other components, and configuration management.

Instantiating and Removing Servlets

After the servlet engine instantiates the servlet, the servlet engine calls the servlet's `init` method to perform any necessary initialization. You can override this method to perform an initialization function for the servlet's life, such as initializing a counter.

When a servlet is removed from service, the servlet engine calls the `destroy` method in the servlet so that the servlet can perform any final tasks and deallocate resources. You can override this method to write log messages or clean up any lingering connections that won't be caught in garbage collection.

Request Handling

When a request is made, the server runtime hands the incoming data to the servlet

engine. The servlet engine processes the request's input data, such as form data, cookies, session information, and URL name-value pairs, into an `HttpServletRequest` request object type.

The servlet engine also creates an `HttpServletResponse` response object type. The engine then passes both as parameters to the servlet's `service` method.

In an HTTP servlet, the default `service` method routes requests to another method based on the HTTP transfer method. For example, HTTP `POST` requests are sent to the `doPost` method, HTTP `GET` requests are sent to the `doGet` method, and so on. This enables the servlet to process request data differently, depending on which transfer method is used.

Since the routing takes place in the `service` method, you generally do not override `service` in an HTTP servlet. Instead, override `doGet`, `doPost`, and so on, depending on the request type you expect.

To perform the tasks to answer a request, override the `service` method for generic servlets, and the `doGet` or `doPost` methods for HTTP servlets. Very often, this means accessing EJB components or CDI beans to perform business operations, then collating the information in the `request` object or in a JDBC `ResultSet` object.

Using Jakarta Server Pages

Payara Server and Micro supports the following features:

- Jakarta Server Pages (JSP) Specification
- Pre-compilation of JSP files, which is especially useful for production servers
- JSP tag libraries and standard portable tags

This section describes how to use Jakarta Server Pages (JSP files) as page templates in a web application.

JSP Tag Libraries and Standard Portable Tags

Payara Server and Micro supports tag libraries and standard portable tags. For more

information, see the Jakarta Standard Tag Library (JSTL) specification.

> **NOTE** The standard tag libraries are automatically available to all web applications.

JSP Caching

JSP caching lets you cache tag invocation results within the servlet engine. Each view can be cached using different cache criteria. For example, suppose you have invocations to view stock quotes, weather information, and so on. The stock quote result can be cached for 10 minutes, the weather report result for 30 minutes, and so on.

For more information about response caching as it pertains to servlets, see Caching Servlet Results.

Enabling JSP Caching

To globally enable JSP caching, set the `jspCachingEnabled` property to `true`. The default is `false`. For example:

```
asadmin set server-config.web-container.property.jspCachingEnabled=true
```

To enable JSP caching for a single web application, follow these steps:

1. Extract the `/META-INF/jspcachtags.tld` file from the `as-install/glassfish/modules/web-glue.jar` file.
2. Create a new JAR file (for example, `jspcachtags.jar`) containing just the `/META-INF/jspcachtags.tld` file previously extracted.
3. Bundle this new JAR file in the `WEB-INF/lib` directory of your web application.

> **NOTE** Web applications that use JSP caching without bundling the tag library are not portable.

Refer to the caching tags in JSP files as follows:

```
<%@ taglib prefix="prefix" uri="http://glassfish.org/taglibs/cache" %>
```

Subsequently, the cache tags are available as `<${prefix}:cache>` and `<${prefix}:flush>`. For example, if your prefix is `mypfx`, the cache tags are available as `<mypfx:cache>` and `<mypfx:flush>`.

Caching Scope

JSP caching is available in three different scopes: `request`, `session`, and `application`. The default is `application`. To use a cache in `request` scope, a web application must configure the `com.sun.appserv.web.taglibs.cache.CacheRequestListener` as a web listener in the application's `web.xml` deployment descriptor, as follows:

```
<listener>
   <listener-class>
com.sun.appserv.web.taglibs.cache.CacheRequestListener</listener-class>
</listener>
```

Likewise, for a web application to utilize a cache in `session` scope, it must configure the `com.sun.appserv.web.taglibs.cache.CacheSessionListener` as a web listener in its `web.xml` deployment descriptor, as follows:

```
<listener>
   <listener-class>
com.sun.appserv.web.taglibs.cache.CacheSessionListener</listener-class>
</listener>
```

To utilize a cache in `application` scope, a web application doesn't need to configure any extra listener. The `com.sun.appserv.web.taglibs.cache.CacheContextListener` is already specified in the `jspcachtags.tld` file.

The `cache` Tag

The `cache` tag caches the body between the beginning and ending tags according to the attributes specified. The first time the tag is encountered, the body content is executed and cached. Each subsequent time it is run, the cached content is checked to

see if it needs to be refreshed and if so, it is executed again, and the cached data is refreshed.

Otherwise, the cached data in the page is served.

Attributes of `cache`

The following table describes attributes for the `cache` tag.

Table 187. The `cache` tag attributes

Attribute	Default	Description
`key`	`ServletPath_Suffix`	(optional) The name used by the container to access the cached entry. The cache key is suffixed to the servlet path to generate a key to access the cached entry.
`timeout`	`60s`	(optional) The time in seconds after which the body of the tag is executed and the cache is refreshed. By default, this value is interpreted in seconds. To specify a different unit of time, add a suffix to the timeout value as follows: `s` for seconds, `m` for minutes, `h` for hours, `d` for days. For example, `2h` specifies two hours.
`nocache`	`false`	(optional) If set to `true`, the body content is executed and served as if there were no `cache` tag. This offers a way to programmatically decide whether the cached response is sent or whether the body has to be executed, though the response is not cached.
`refresh`	`false`	(optional) If set to `true`, the body content is executed and the response is cached again. This lets you programmatically refresh the cache immediately regardless of the `timeout` setting.
`scope`	`application`	(optional) The scope of the cache. Can be `request`, `session`, or `application`. See Caching Scope.

Example of `cache` Tag Annotation Usage

The following example represents a cached JSP file:

```
<%@ taglib prefix="mypfx" uri="http://glassfish.org/taglibs/cache" %>
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
<mypfx:cache key="${sessionScope.loginId}" nocache="${param.nocache}"
refresh="${param.refresh}" timeout="10m">
    <c:choose>
        <c:when test="${param.page == 'frontPage'}">
            <%-- get headlines from database --%>
        </c:when>
        <c:otherwise>
            ...
        </c:otherwise>
    </c:choose>
</mypfx:cache>
<mypfx:cache timeout="1h">
    <h2>Local News</h2>
    <%-- get the headline news and cache them --%>
</mypfx:cache>
```

The `flush` Tag

Forces the cache to be flushed. If a `key` is specified, only the entry with that key is flushed. If no key is specified, the entire cache is flushed.

Attributes of `flush`

The following table describes the attributes used for the `flush` tag.

Table 188. The `flush` tag attributes

Attribute	Default	Description
`key`	`ServletPath_Suffix`	(optional) The name used by the container to access the cached entry. The cache key is suffixed to the servlet path to generate a key to access the cached entry. If no key is specified, a number is generated according to the position of the tag in the page.
`scope`	`application`	(optional) The scope of the cache. Can be `request`, `session`, or `application`. See Caching Scope

Examples of `flush`

To flush the entry with `key="foobar"`:

```
<mypfx:flush key="foobar"/>
```

To flush the entire cache:

```
<c:if test="${empty sessionScope.clearCache}">
   <mypfx:flush />
</c:if>
```

Options for Compiling JSP Files

Payara Server provides the following ways of compiling JSP source files into servlets:

- JSP files are automatically compiled at runtime.
- The `asadmin deploy` command has a `--precompilejsp` option.
- The `jspc` command line tool allows you to precompile JSP files at the command line.

Creating and Managing Sessions

This section describes how to create and manage HTTP sessions that allows users and transaction information to persist between interactions.

Configuring Sessions

HTTP Sessions, Cookies, and URL Rewriting

To configure whether and how HTTP sessions use cookies and URL rewriting, edit the `session-properties` and `cookie-properties elements in the` `glassfish-web.xml` or `payara-web.xml` file for an individual web application. For more details about the properties you can configure, see "session-properties" "cookie-properties" and in the Payara Server Application Deployment section.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

For information about configuring default session properties for the entire web

container, see Using the `default-web.xml` File.

Saving Sessions During Redeployment

Whenever an application redeployment is done, the sessions at that transit time become invalid unless you use the `--keepstate=true` option of the `asadmin redeploy` command. For example:

```
asadmin redeploy --keepstate=true --name hello.war
```

The default for the `--keepstate` is `false`. This option is supported only on the default server instance, named `server`.

NOTE	This option is not supported and ignored for other targets (clusters and deployment groups included).

For web applications, this feature is applicable only if in the `payara-web.xml` / `glassfish-web.xml` deployment descriptor files when the `persistence-type` attribute of the `session-manager` element is set to `file`.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

If any active web session fails to be preserved or restored, none of the sessions will be available when the redeployment is complete. However, the redeployment continues and a warning is logged.

The new class loader of the redeployed application is used to deserialize any sessions previously saved. The usual restrictions about serialization and deserialization apply. For example, any application-specific class referenced by a session attribute may evolve only in a backward-compatible fashion.

For more information about class loaders, see "Class Loaders".

Logging Session Attributes

You can write specific session attribute values to the HTTP service access log. The access log format placeholder `%session.name%` logs one of the following:

- The value of the session attribute with the `${ATTRIBUTE_NAME}-name` format.
- `NULL-SESSION-ATTRIBUTE-name` if the named attribute does not exist in the session.
- `NULL-SESSION` if no session exists.

Distributed Sessions and Persistence

A distributed HTTP session can run in multiple Payara Server/Micro instances, provided the following criteria are met:

- Each server instance has the same distributable web application deployed to it. The `web-app` element of the `web.xml` deployment descriptor file must have the `distributable` sub-element specified.
- The web application uses high-availability session persistence. If a non-distributable web application is configured to use high-availability session persistence, a warning is written to the server log, and the session persistence type reverts to `memory`. See The `memory` persistence Type.
- All objects bound into a distributed session must be of the types listed in the table below.
- The web application must be deployed using the `deploy` command with the `--availabilityenabled` option set to `true`.

NOTE	Contrary to what is recommended in the Jakarta Servlet specification, Payara Server does not throw an `IllegalArgumentException` if an object type not supported for failover is bound into a distributed session.

Keep the distributed session size as small as possible. Session size has a direct impact on overall system throughput.

In the event of an instance or hardware failure, another server instance can take over a distributed session, with the following limitations:

- If a distributable web application references a Jakarta EE component or resource, the reference might be lost. See the table below for a list of the types of references that `HTTPSession` failover supports.
- References to open files or network connections are lost.

In the following table, **No** indicates that failover for the object type might not work in all cases and that no failover support is provided. However, failover might work in some cases for that object type. For example, failover might work because the class implementing that type is serializable.

For more information about the `InitialContext`, see accessing-the-naming-context.

For more information about transaction recovery, see using-the-transaction-service. For more information about Administered Objects, see "administering-jms-physical-destinations" in the Payara Server General Administration section.

Table 189. Object Types Supported for Web Application Session State Failover

Java Object Type	Failover Support
Co-located or distributed stateless session, stateful session, or entity bean references	Yes
JNDI context	Yes, for `InitialContext` and `java:comp/env` namespace references
`UserTransaction`	Yes, but if the instance that fails is never restarted, any prepared global transactions are lost and might not be correctly rolled back or committed.
JDBC DataSource	No
Jakarta Messaging (JMS) ConnectionFactory and Destination	No
Jakarta Mail Session	No

Java Object Type	Failover Support
Connection Factory	No
Administered Object	No
SOAP Web Service reference	No
Serializable Java types	Yes
Extended persistence context	No

Session Managers

A session manager automatically creates new session objects whenever a new session starts. In some circumstances, clients do not join the session, for example, if the session manager uses cookies and the client does not accept cookies.

Azul Payara offers these session management options, determined by the `session-manager` element's `persistence-type` attribute in the `glassfish-web.xml` / `payara-web.xml` file:

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

- The `memory` Persistence Type
- The `file` Persistence Type, which uses a file to store session data
- The `hazelcast` Persistence Type, which uses the Data Grid to store session data

NOTE	If the session manager configuration contains an error, the error is written to the server log and the default (`memory`) configuration is used.

For more information, see "session-manager" in the Payara Server Application Deployment section.

The `memory` Persistence Type

This persistence type is not designed for a production environment that requires session persistence as it provides no actual session persistence. However, you can

configure it so that the session state in memory is written to the file system prior to server shutdown.

To specify the `memory` persistence type for a specific web application, edit the `glassfish-web.xml` or `payara-web.xml` file as in the following example. The `persistence-type` attribute is optional, but must be set to `memory` if included. This overrides the web container availability settings for the web application.

```
<payara-web-app>
 <session-config>
     <session-manager persistence-type="memory">
         <manager-properties>
             <property name="sessionFilename" value="sessionstate" />
         </manager-properties>
     </session-manager>
 </session-config>
</payara-web-app>
```

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

The only manager property that the `memory` persistence type supports is `sessionFilename`, which is listed under "manager-properties" in the Payara Server Application Deployment section.

The `sessionFilename` property specifies the name of the file where sessions are serialized and persisted if the web application or the server is stopped. To disable this behavior, specify an empty string as the value of `sessionFilename`. The default value is an empty string.

The `file` Persistence Type

This persistence type provides session persistence to the local file system, and allows a single server domain to recover the session state after a failure and restart. The session state is persisted in the background, and the rate at which this occurs is configurable.

The store also provides passivation and activation of the session state to help control the amount of memory used. This option is not supported in a production environment

since it is not adequate for any high-availability environments. However, it is useful for a development system with a single server instance.

NOTE	Make sure the `delete` option is set in the `server.policy` file, or expired file-based sessions might not be deleted properly. For more information about `server.policy`, see the-server.policy-file.

To specify the `file` persistence type for a specific web application, edit the `glassfish-web.xml` or `payara-web.xml` file as in the following example. Note that `persistence-type` must be set to `file`. This overrides the web container availability settings for the web application.

```
<payara-web-app>
  <session-config>
      <session-manager persistence-type="file">
          <store-properties>
              <property name="directory" value="sessiondir" />
          </store-properties>
      </session-manager>
  </session-config>
</payara-web-app>
```

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

The `file` persistence type supports all the manager properties listed under "manager-properties" in the Payara Server Application Deployment section except `sessionFilename`, and supports the `directory` store property listed under "store-properties" in the same document.

The `hazelcast` Persistence Type

The `hazelcast` persistence type uses Payara Server/Micro Data Grid to store information in the internal distributed memory grid. All server instances that join the data grid (regardless of the deployment groups they belong to) will store partitioned information of the session data courtesy of Hazelcast. This allows sessions to be distributed properly with failover in place. For details, see Distributed Sessions and

Persistence. This is the default persistence type for both Payara Server and Micro instances.

IMPORTANT	Hazelcast and the Data Grid must be enabled for the Payara Server domain or a corresponding Payara Micro instance for this persistence type to work.
WARNING	Session data will only be stored in non-lite Data Grid members, so for this persistence type to work properly, the Data Grid must be conformed by more than **1 non-lite member**, so that data is stored in the grid in a resilient manner.

To use the `hazelcast` persistence type, you must enable availability first. Select the *Availability Service* component under the relevant configuration in the Administration Console. Check the *Availability Service* box. To enable availability for the web container, select the *Web Container Availability* tab, then check the *Availability Service* box.

All instances in a cluster or deployment group should have the same availability settings to ensure consistent behavior.

Using Comet

This section explains the Comet programming technique and how to create and deploy a Comet-enabled application in Payara Server or Micro.

Introduction to Comet

Comet is a programming technique that allows a web server to send updates to clients without requiring the clients to explicitly request them.

This kind of programming technique is called **server push**, which means that the server pushes data to the client. The opposite style is **client pull**, which means that the client must pull the data from the server, usually through a user-initiated event, such as a button click.

Web applications that use the Comet technique can deliver updates to clients in a more

timely manner than those that use the client-pull style while avoiding the latency that results from clients frequently polling the server.

One of the more common use cases for Comet is a chat room application. When the server receives a message from one of the chat clients, it needs to send the message to the other clients without requiring them to ask for it.

With Comet, the server can deliver messages to the clients as they are posted rather than expecting the clients to poll the server for new messages.

To accomplish this scenario, a Comet application establishes a long-lived HTTP connection. This connection is suspended on the server side, waiting for an event to happen before resuming. This kind of connection remains open, allowing an application that uses the Comet technique to send updates to clients when they are available rather than expecting clients to reopen the connection to poll the server for updates.

The Grizzly Implementation of Comet

A limitation of the Comet technique is that you must use it with a web server that supports non-blocking connections to avoid poor performance. Non-blocking connections are those that do not need to allocate one thread for each request. If the web server were to use blocking connections then it might end up holding many thousands of threads, thereby hindering its scalability.

Azul Payara includes the Grizzly HTTP Engine, which enables asynchronous request processing (ARP) by avoiding blocking connections. Grizzly's ARP implementation accomplishes this by using the Java NIO API.

With Java NIO, Grizzly enables greater performance and scalability by avoiding the limitations experienced by traditional web servers that must run a thread for each request. Instead, Grizzly's ARP mechanism makes efficient use of a thread pool system and also keeps the state of requests so that it can keep requests alive without holding a single thread for each of them.

Client Technologies to Use with Comet

In addition to creating a web component that uses the Comet APIs, you need to enable your client to accept asynchronous updates from the web component. To accomplish this, you can use Iframes, plain JavaScript, or a robust JavaScript framework, such as the Dojo Toolkit.

The next section explains the two kinds of connections that you can make to the server. While you can use any of the client technologies listed in this section with either kind of connection, it is more challenging to use JavaScript with an HTTP-streaming connection.

Types of Comet Connections

When working with Comet, as implemented in Grizzly, you have two different ways to handle client connections to the server:

HTTP Streaming

The HTTP Streaming technique keeps a connection open indefinitely. It never closes, even after the server pushes data to the client.

In the case of HTTP streaming, the application sends a single request and receives responses as they come, reusing the same connection forever. This technique significantly reduces the network latency because the client and the server don't need to open and close the connection.

The basic life cycle of an application using HTTP-streaming is:

> Request Suspend Data available Write response Data available Write response

The client makes an initial request and then suspends the request, meaning that it waits for a response. Whenever new data is available, the server writes it to the response.

Long Polling

The long-polling technique is a combination of server-push and client-pull because the

client needs to resume the connection after a certain amount of time or after the server pushes an update to the client.

The basic life cycle of an application using long-polling is:

> Request → Suspend → Data available → Write response → Resume

The client makes an initial request and then suspends the request. When an update is available, the server writes it to the response. The connection closes, and the client optionally resumes the connection.

How to Choose the Type of Connection

If you anticipate that your web application will need to send frequent updates to the client, you should use the HTTP-streaming connection so that the client does not have to frequently reestablish a connection. If you anticipate less frequent updates, you should use the long-polling connection so that the web server does not need to keep a connection open when no updates are occurring.

One caveat to using the HTTP-streaming connection is that if you are streaming through a proxy, the proxy can buffer the response from the server. So, be sure to test your application if you plan to use HTTP-streaming behind a proxy.

Grizzly Comet

For details on using Grizzly Comet including a sample application, refer to the Grizzly Comet documentation on GitHub (`https://javaee.github.io/grizzly/comet.html`).

Grizzly's support for Comet includes a small set of APIs that make it easy to add Comet functionality to your web applications. The Grizzly Comet APIs that developers use most often are the following:

- `CometContext`: A Comet context, which is a shareable space to which applications subscribe to receive updates.
- `CometEngine`: The entry point to any component using Comet. Components can be

servlets, JavaServer Pages (JSP), JavaServer Faces components, or pure Java classes.

- `CometEvent`: Contains the state of the `CometContext` object
- `CometHandler`: The interface an application implements to be part of one or more Comet contexts.

The way a developer would use this API in a web component is to perform the following tasks:

1. Register the context path of the application with the `CometContext` object:

```
var cometEngine = CometEngine.getEngine();
var cometContext = cometEngine.register(contextPath)
```

2. Register the `CometHandler` implementation with the `CometContext` object:

```
cometContext.addCometHandler(handler);
```

3. Notify one or more `CometHandler` implementations when an event happens:

```
cometContext.notify(handler);
```

Enabling Comet

Before running a Comet-enabled application, you need to enable Comet in the HTTP listener for your application by setting a special attribute in the associated protocol configuration. The following example shows the `asadmin set` command that adds this attribute:

```
asadmin set server-config.network-config.protocols.protocol.${protocol-name
}.http.comet-support-enabled="true"
```

Advanced Web Application Features

The Server's Default Locale

To set the default locale of the entire server runtime, which determines the locale of the Payara Server Administration Console (In a multi-language distribution variant), text

entries in the logs, and so on, use the Administration Console. Select the *domain* component.

Then type a value in the *Locale* field. Values must conform to a valid `java.util.Locale` constant (see the Javadoc for the Locale class for more information).

Servlet Character Encoding

This section explains how the server runtime determines the character encoding for the servlet request and response. Valid encoding formats must conform to the list of Java's supported encodings.

Servlet Request

When processing a servlet request, the server uses the following order of precedence, first to last, to determine the request character encoding:

- The `getCharacterEncoding` method.
- A hidden field in the form, specified by the `form-hint-field` attribute of the `parameter-encoding` element in the `glassfish-web.xml` or `payara-web.xml` deployment descriptor
- The `default-charset` attribute of the `parameter-encoding` element in the `glassfish-web.xml` or `payara-web.xml` deployment descriptor
- The default encoding, which is `ISO-8859-1`

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

For details about the `parameter-encoding` element, see "parameter-encoding" in the Payara Server Application Deployment section.

Servlet Response

When processing a servlet response, the server uses the following order of precedence, first to last, to determine the response character encoding:

- The `setCharacterEncoding` or `setContentType` method
- The `setLocale` method
- The default encoding, which is `ISO-8859-1`

Virtual Server Properties

You can set virtual server properties in the following ways:

- You can define virtual server properties using the `asadmin create-virtual-server` command. For example:

  ```
  asadmin create-virtual-server --hosts localhost --property authRealm=ldap MyVS
  ```

  For more details and a complete list of virtual server properties, see "`create-virtual-server`".

- You can define virtual server properties using the `asadmin set` command.

  For example:

  ```
  asadmin set server-config.http-service.virtual-server.MyVS.property.authRealm
  ="ldap"
  ```

- You can define virtual server properties using the Administration Console. Select the *HTTP Service* component under the relevant configuration, then select *Virtual Servers*, and select the desired virtual server. Select *Add Property*, enter the property name and value, check the *Enable* box, and save your changes.

Using the `default-web.xml` File

You can use the `default-web.xml` file to define features such as filters and security constraints that apply to all web applications deployed in a Payara Server domain.

For example, directory listings are disabled by default for added security. To enable directory listings, in your domain's `default-web.xml` file, search for the definition of the servlet whose `servlet-name` is equal to `default`, and set the value of the `init-param` named `listings` to `true`.

Then redeploy your web application if it has already been deployed, or restart the server. Here's an excerpt of the file configured as stated:

```
<init-param>
   <param-name>listings</param-name>
   <param-value>true</param-value>
</init-param>
```

If `listings` is set to `true`, you can also determine how directory listings are sorted. Set the value of the `sortedBy` parameter to `NAME`, `SIZE`, or `LAST_MODIFIED`. Then redeploy your web application if it has already been deployed, or restart the server instance.

Here's an excerpt of the file configured as stated:

```
<init-param>
   <param-name>sortedBy</param-name>
   <param-value>LAST_MODIFIED</param-value>
</init-param>
```

The `mime-mapping` elements in `default-web.xml` are global and inherited by all web applications. You can override these mappings or define your own using `mime-mapping` elements in your web application's `web.xml` file. For more information about `mime-mapping` elements, see the Jakarta Servlet specification.

To Use the `default-web.xml` File

1. Place the JAR file for the filter, security constraint, or any other specific web component features in the `domain-dir/lib` directory.
2. Edit the `domain-dir/config/default-web.xml` file to refer to the JAR file.
3. Restart the domain and any server instances.

Configuring Logging and Monitoring in the Web Container

Select *Logger Settings* under the relevant configuration, or select the *Instances* component, select the instance from the table, and select the *Monitor* tab.

Catalina Specific Features

IMPORTANT	**Catalina** is the internal implementation of the Servlet implementation used in Azul Payara, originated from Apache Tomcat's own implementation. Catalina will be replaced in future releases and as such the specifics documented here may change or be completely rewritten.

Configuring Valves and Catalina Listeners

You can configure custom valves and **Catalina** listeners for web modules or virtual servers by defining properties. A valve class must implement the `org.glassfish.web.valve.GlassFishValve` interface. A listener class for a virtual server must implement the `org.apache.catalina.ContainerListener` or `org.apache.catalina.LifecycleListener` interfaces.

A listener class for a web module must implement one the following interfaces:

- `org.apache.catalina.ContainerListener`
- `org.apache.catalina.LifecycleListener`
- `org.apache.catalina.InstanceListener`

In the `glassfish-web.xml`/`payara-web.xml` deployment descriptor, valve and listener properties for a web module look like this:

```
<payara-web-app>
   <property name="valve_1" value="org.glassfish.extension.Valve"/>
   <property name="listener_1" value=
"org.glassfish.extension.MyLifecycleListener"/>
</payara-web-app>
```

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

You can define these same properties for a virtual server configuration. For more information, see Virtual Server Properties.

Using a `context.xml` File

You can define an Apache Tomcat `context.xml` configuration file for all web applications, for web applications assigned to a specific virtual server, or for a specific web application.

To define a global `context.xml` file, place the file in the `domain-dir/config` directory and name it `context.xml`.

Use the `contextXmlDefault` property to specify the name and the location, relative to the domain directory, of the `context.xml` file for a specific virtual server. Specify this property in one of the following ways:

- In the Administration Console, open the *HTTP Service* component under the relevant configuration. Open the *Virtual Servers* component and scroll down to the bottom of the page.

  Enter `contextXmlDefault` as the property name and the path and file name relative to domain-dir as the property value.

- Use the `asadmin create-virtual-server` command. For example:

```
asadmin create-virtual-server --property contextXmlDefault=config/vs1ctx.xml vs1
```

- Use the `asadmin set` command for an existing virtual server. For example:

```
asadmin set server-config.http-service.virtual-
server.vs1.property.contextXmlDefault=config/myctx.xml
```

To define a `context.xml` file for a specific web application, place the file in the `META-INF` directory of the corresponding application and name it `context.xml`.

For more information about virtual server properties, see Virtual Server Properties. For more information about the `context.xml` file, see The Context Container.

NOTE	Not all context definitions are supported by Azul Payara.

Enabling Webdav

To enable WebDav in Payara Server, you edit the `web.xml` and `glassfish-web.xml` / `payara-web.xml` files as follows.

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

First, enable the WebDav servlet in your `web.xml` file:

```
<servlet>
   <servlet-name>webdav</servlet-name>
   <servlet-class>org.apache.catalina.servlets.WebdavServlet</servlet-class>
   <init-param>
      <param-name>debug</param-name>
      <param-value>0</param-value>
   </init-param>
   <init-param>
      <param-name>listings</param-name>
      <param-value>true</param-value>
   </init-param>
   <init-param>
      <param-name>readonly</param-name>
      <param-value>false</param-value>
   </init-param>
</servlet>
```

Then define the servlet mapping associated with your WebDav servlet in your `web.xml` file:

```
<servlet-mapping>
   <servlet-name>webdav</servlet-name>
   <url-pattern>/webdav/*</url-pattern>
</servlet-mapping>
```

To protect the WebDav servlet so other users can't modify it, you must add a security constraint in your `web.xml` deployment descriptor:

```
<security-constraint>
   <web-resource-collection>
      <web-resource-name>Login Resources</web-resource-name>
      <url-pattern>/webdav/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>Admin</role-name>
   </auth-constraint>
   <user-data-constraint>
```

```
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>default</realm-name>
    </login-config>
    <security-role>
        <role-name>Admin</role-name>
    </security-role>
</security-constraint>
```

Then define a security role mapping in your `glassfish-web.xml` or `payara-web.xml` deployment descriptor:

```
<security-role-mapping>
    <role-name>Admin</role-name>
    <group-name>Admin</group-name>
</security-role-mapping>
```

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

Enable the security manager as described in enabling-and-disabling-the-security-manager.

You can now use any WebDav client by connecting to the WebDav servlet URL, which has this format:

```
http://${host}:${port}/${context-root}/webdav/file
```

For example:

```
http://localhost:80/myapp-webdav/webdav/index.html
```

You can add the WebDav servlet to your `default-web.xml` file to enable it for all applications, but you can't set up a security role mapping to protect it.

Using SSI

To enable SSI (server-side includes) processing for a specific web module, add the `SSIServlet` to your `web.xml` deployment descriptor as follows:

```
<web-app>
   <servlet>
      <servlet-name>ssi</servlet-name>
      <servlet-class>org.apache.catalina.ssi.SSIServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>ssi</servlet-name>
      <url-pattern>*.shtml</url-pattern>
   </servlet-mapping>
   <mime-mapping>
      <extension>shtml</extension>
      <mime-type>text/html</mime-type>
   </mime-mapping>
</web-app>
```

To enable SSI processing for all web modules, un-comment the corresponding sections in the `default-web.xml` file.

If the `mime-mapping` is not specified in `web.xml`, Payara Server attempts to determine the MIME type from `default-web.xml` or the operating system default.

You can configure the following `init-param` values for the `SSIServlet`.

Table 190. `SSIServlet init-param` *Values*

init-param	Type	Default	Description
`buffered`	`boolean`	`false`	Specifies whether the output should be buffered.
`debug`	`int`	`0` (for no debugging)	Specifies the debugging level.
`expires`	`Long`	`Expires` header in HTTP response not set	Specifies the expiration time in seconds.
`inputEncoding`	`String`	OS encoding	Specifies encoding for the SSI input if there is no URL content encoding specified.
`isVirtualWebappRelative`	`boolean`	`false` (relative to the given SSI file)	Specifies whether the virtual path of the `#include` directive is relative to the `content-root`.
`outputEncoding`	`String`	`UTF-8`	Specifies encoding for the SSI output.

For more information about SSI, see `http://httpd.apache.org/docs/2.2/mod/mod_include.html`.

Using CGI

To enable CGI (common gateway interface) processing for a specific web module, add the `CGIServlet` to your `web.xml` file as follows:

```
<web-app>
   <servlet>
      <servlet-name>cgi</servlet-name>
      <servlet-class>org.apache.catalina.servlets.CGIServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>cgi</servlet-name>
      <url-pattern>/cgi-bin/*</url-pattern>
   </servlet-mapping>
</web-app>
```

To enable CGI processing for all web modules, un-comment the corresponding sections in the `default-web.xml` file.

Package the CGI program under the `cgiPathPrefix`. The default `cgiPathPrefix` is `WEB-INF/cgi`. For security, it is highly recommended that the contents and binaries of CGI programs be prohibited from direct viewing or download.

For information about hiding directory listings, see Using the `default-web.xml` File.

Invoke the CGI program using a URL of the following format:

```
http://${host}:${port}/${context-root}/cgi-bin/cgi-name
```

For example:

```
http://localhost:8080/mycontext/cgi-bin/hello
```

You can configure the following `init-param` values for the `CGIServlet`.

Table 191. `CGIServlet init-param` Values

Parameter	Type	Default	Description
`cgiPathPrefix`	`String`	`WEB-INF/cgi`	Specifies the subdirectory containing the CGI programs.
`debug`	`int`	`0` (for no debugging)	Specifies the debugging level.
`executable`	`String`	`perl`	Specifies the executable for running the CGI script.
`parameterEncoding`	`String`	`System.getProperty("file.encoding", "UTF-8")`	Specifies the parameter's encoding.
`passShellEnvironment`	`boolean`	`false`	Specifies whether to pass shell environment properties to the CGI program.

To work with a native executable, do the following:

1. Set the value of the `init-param` named executable to an empty `String` in the `web.xml` file.
2. Make sure the executable has its executable bits set correctly.
3. Use directory deployment to deploy the web module. Do not deploy it as a WAR file, because the executable bit information is lost during the process of `jar` and `unjar`.

Alternate Document Roots

An alternate document root (`docroot`) allows a web application to serve requests for certain resources from outside its own document root, based on whether those requests match one (or more) of the URI patterns of the web application's alternate document roots.

To specify an alternate document root for a web application or a virtual server, use the `alternatedocroot_n` property, where n is a positive integer greater than zero that allows specifying more than one.

This property can be a sub-element of a `glassfish-web-app`/`payara-web-app` element in the `glassfish-web.xml`/`payara-web.xml` file or a virtual server property.

For more information about this element, see "payara-web-app" in the Payara Server Application Deployment section.

A virtual server's alternate document roots are considered only if a request does not map to any of the web modules deployed on that virtual server. A web module's alternate document roots are considered only once a request has been mapped to that web module.

If a request matches an alternate document root's URI pattern, it is mapped to the alternate document root by appending the request URI (minus the web application's context root) to the alternate document root's physical location (directory). If a request matches multiple URI patterns, the alternate document root is determined according to the following precedence order:

- Exact match
- Longest path match
- Extension match

For example, the following properties specify three `payara-web.xml` document root configurations. The URI pattern of the first alternate document root uses an exact match, whereas the URI patterns of the second and third alternate document roots use extension and longest path prefix matches, respectively:

```
<payara-web-app>
    <property name="alternatedocroot_1" value="from=/my.jpg dir=/srv/images/jpg"/>
    <property name="alternatedocroot_2" value="from=*.jpg dir=/srv/images/jpg"/>
    <property name="alternatedocroot_3" value="from=/jpg/* dir=/src/images"/>
</payara-web-app>
```

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

The `value` of each alternate document root has two components: The first component, `from`, specifies the alternate document root's URI pattern, and the second component, `dir`, specifies the alternate document root's physical location (directory).

Suppose the above examples belong to a web application deployed at `http://examples.payara.fish/myapp`. The first alternate document root maps any requests with this URL:

```
http://examples.payara.fish/myapp/my.jpg
```

To this resource:

```
/svr/images/jpg/my.jpg
```

The second alternate document root maps any requests with a `*.jpg` suffix, such as:

```
http://examples.payara.fish/myapp/*.jpg
```

To this physical location:

```
/svr/images/jpg
```

The third alternate document root maps any requests whose URI starts with `/myapp/jpg/`, such as:

```
http://examples.payara.fish/myapp/jpg/*
```

To the same directory as the second alternate document root.

For example, the second alternate document root maps this request:

```
http://examples.payara.fish/myapp/abc/def/my.jpg
```

To:

```
/srv/images/jpg/abc/def/my.jpg
```

The third alternate document root maps:

```
http://examples.payara.fish/myapp/jpg/abc/resource
```

To:

```
/srv/images/jpg/abc/resource
```

If a request does not match any of the target web application's alternate document roots, or if the target web application does not specify any alternate document roots, the request is served from the web application's standard document root, as usual.

Using Jakarta Faces Technology

This chapter describes how to use the Jakarta Faces (JSF) API.

Introducing Jakarta Faces

Jakarta Faces is a server-side component framework for creating diverse Java technology–based web applications for browser-focused environments.

Jakarta Faces consists of the following:

- An API for representing components and managing their state; handling events, server-side validation, and data conversion; defining page navigation; supporting internationalization and accessibility
- Tag library sets for adding components to web pages and for linking components to server-side objects.

Jakarta Faces provides a well-defined programming model and various **tag libraries**. These tag libraries contain **tag handlers** that implement component tags. These features significantly ease the burden of creating and maintaining web applications with server-side user interfaces.

Jakarta Faces Applications

The functionality provided by a Jakarta Faces application is similar to that of any other Java web-based application. A typical Jakarta Faces application includes the following parts.

- A set of web pages in which components are rendered at runtime.
- A set of tags to add components to the web page.
- A set of managed Java beans, which are lightweight, container-managed objects (POJOs).
    - In a Jakarta Faces application, managed beans usually serve as **backing beans**, which define properties to store data and methods and event handlers for UI components on a page.
- A web deployment descriptor (`web.xml`).
- *(Optional)* One or more application configuration resource files, such as a `faces-config.xml` file, which can be used to define page navigation rules and provide further customization options for the application.
- *(Optional)* A set of custom objects, which can include custom components, validators, converters, or listeners.
- *(Optional)* A set of custom tags for representing custom or complex objects on the page.

Configurable JSF Options

This section documents specific configuration settings that are not available in the standard deployment descriptors or configuration files described above as part of the Jakarta Faces API.

Parallel Initialization

It is possible to specify an Azul Payara specific JSF context parameter in the `web.xml` deployment descriptor, named `fish.payara.faces.enableParallelInit` to provide parallel initialization benefits to the application in question.

Usage

When the `fish.payara.faces.enableParallelInit` context parameter is set to `true` some of the internal initialisation of Mojarra (the JSF implementation Payara Server uses) is performed in parallel.

To this effect, this the platform fork/join thread pool is used, which is the same pool used for processing JDK 8 Streams in parallel.

Example

Here's an example that sets the option to `true` in `web.xml`:

```
<context-param>
    <param-name>fish.payara.faces.enableParallelInit</param-name>
    <param-value>true</param-value>
</context-param>
```

The `fish.payara.faces.enableParallelInit` context parameter can also be combined with the `com.sun.faces.enableThreading` parameter:

```
<context-param>
    <param-name>com.sun.faces.enableThreading</param-name>
    <param-value>true</param-value>
</context-param>
```

This is an existing Mojarra-specific parameter to also enable parallel initialisation, but in a slightly earlier stage of the initialisation process.

IMPORTANT	As an Azul Payara specific implementation this will use the default Platform Concurrency executor if there is one, if not it will create a single use executor with 5 threads.

Using an Alternative Jakarta Faces Implementation

As a special case of how the classloader hierarchy can be modified (see configuring-delegation), Azul Payara allows the use of a Jakarta Faces implementation that is bundled on a web application.

This feature may be useful to switch to a different implementation than the default one

included with the server.

To configure this feature, you need to indicate within the `payara-web.xml` / `glassfish-web.xml` deployment descriptor that the server should use a bundled Jakarta Faces implementation as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-web-app PUBLIC "-//Payara.fish//DTD Payara Server 7 Servlet 6.1
Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-web-app_6_1-1.dtd">
<payara-web-app error-url="">
    <class-loader delegate="false"/>
    <property name="useBundledJsf" value="true" />
</payara-web-app>
```

IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

By specifying these options, the bundled Jakarta Faces implementation within your web application will be used instead of the default implementation included within the server.

IMPORTANT	If deploying a EAR application you must bundle the Jakarta Faces implementation library within the embedded WAR component. You cannot add an alternative Jakarta Faces implementation as an EAR library.

Using Jakarta MVC

This chapter describes how to use the Jakarta MVC API, an action-based framework for building web applications using the model view controller architecture pattern.

The model, or the **M**, refers to the underlying application data. In Jakarta EE, this is generally a Jakarta Persistence entity or its projected instance (DTOs etc).

The view or **V**, refers to the presentation of the model data to the user. Depending on the framework, there are different view technologies that can be used. For instance Jakarta

Faces uses Facelets as its view technology, while Jakarta MVC supports both Facelets and Jakarta Server Pages (which is the default).

The controller, or **C**, refers to the unit that manages user input, connects that input to the model and returns output to the user. The returned output could be another view, a redirect, a file download, or anything at all, depending on the application. You can think of the controller as the central, coordinating piece of the application that liaises user actions to different parts of the application.

Introducing Jakarta MVC

Jakarta EE has action and component based frameworks for building web applications using the model view controller architecture pattern. Jakarta Faces is a component based framework while Jakarta MVC is an action based one.

An action based web framework is one in which the application code creates an explicit controller that accepts requests and maps them to actions.

Component based web frameworks on the other hand, have the controllers owned and managed by the framework itself. The controller is transparent to the developer in a component based framework.

What Is Jakarta MVC?

Jakarta MVC is an action based web application development framework built on top of Jakarta REST. It is an alternative way to build traditional web applications on the Jakarta EE Platform. Jakarta MVC is an optional standalone specification that is not part of Jakarta EE by default.

As a Jakarta REST based framework, it makes all the features and options for developing REST services available for developing much more traditional web applications.

Action vs Component Based Frameworks

An action based web framework is one in which the application code creates an explicit controller that accepts requests and maps them to actions.

Component based web frameworks on the hand, have the controllers owned and managed by the framework itself. The controller is transparent to the developer in a component based framework.

Dependencies Setup

To start with Jakarta MVC, you will need to add the specification API to your application dependencies. The current release can be obtained from the Payara API BOM. This setup is shown below.

Jakarta MVC Maven Dependencies

```
<project>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>fish.payara.api</groupId>
                <artifactId>payara-bom</artifactId>
                <version>7.2.0</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>jakarta.platform</groupId>
            <artifactId>jakarta.jakartaee-web-api</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>jakarta.mvc</groupId>
            <artifactId>jakarta.mvc-api</artifactId>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

The Controller

A typical Jakarta MVC application has three parts - the view, the controller and a model. The model doesn't have to necessarily be a database entity.

Any data that the UI displays can be considered as the model. With your above setup, a typical "*Hello, World*" will look as follows.

```
@Path("app")
```

```
@Controller
public class AppController {

    @Inject
    Models models;

    @GET
    public String sayHello() {
        models.put("greet", "Hello, World! Jakarta MVC");
        models.put("platform", "Jakarta EE 11 on Azul Payara Community 7");
        models.put("date",  LocalDateTime.now(ZoneOffset.UTC).format
(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")));
        models.put("message", "Your Jakarta MVC application is running!");

        return "greet.xhtml";
    }

}
```

The AppController is a Jakarta REST resource class, identified as such by the `@Path` annotation.nThe `@Controller` annotation is from Jakarta MVC, marking all methods in this class as controller methods.

A controller method is one that returns a view in the form of a string, as done by the `sayHello()` method, or a `jakarta.ws.rs.core.Response` object that wraps a string object, resolvable to a view.

The `sayHello()` method creates the models for the UI by using the `jakarta.mvc.Models` map.

The `Models` are injected into the controller and then the method populates it accordingly with data for the UI to display. In this example, the injected `Models` instance, `models`, is populated with a few strings identified by their keys.

`Models` is essentially a map that makes its values available to the UI through its keys. All of this is done automatically on your behalf by the Jakarta MVC runtime.

The last line of the `sayHello()` method returns `greet.xhtml`. This is the UI that will be rendered when the user navigates to the `sayHello()` method.

For the sample application, the full path is `http://localhost:8080/jakarta-mvc/mvc/app/`, where `jakarta-mvc` is the context path, `mvc` is the Jakarta REST root

resource path, and `app` is the path to the `sayHello()` method.

```
@ApplicationPath("mvc")
public class RestConfiguration extends Application {

}
```

The `Models` map instance is automatically available to the view and as such it can access all the values we put into it.

The View

There are two view technology options you can use with Jakarta MVC. The first and default is the Jakarta Server Pages, or optionally Jakarta Faces.

As Jakarta Server Pages is no longer a popular technology, this chapter will use the much more popular Jakarta Faces option.

The easiest way to tell Jakarta MVC to use Jakarta Faces as the default view technology is to create an empty `faces-config.xml` in the `WEB-INF` folder.

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config xmlns="https://jakarta.ee/xml/ns/jakartaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-facesconfig_4_0.xsd"
              version="4.0">
</faces-config>
```

With that done, the `greet.xhtml` facelet is shown next.

This should be created under a `views` directory in the `WEB-INF` folder.

```
<!DOCTYPE html>
<html lang="en" xmlns:h="http://xmlns.jcp.org/jsf/html">

<h:head>
    <title>Jakarta MVC</title>
</h:head>

<h:body>
    <h1>#{greet}</h1>
    <p>#{message}</p>
    <p>This application is running on #{platform}, deployed on #{date}</p>
</h:body>
```

```
</html>
```

The `greet.xhtml` view is a very simple facelet file that is accessing the models to display to the user. The models that were put in the `Models` map instance are being accessed through the `\#{}` expression, using the key of each value.

For instance the `#{greet}` will return "Hello, World! Jakarta MVC", as was put in the map.

Accessing `http://localhost:8080/jakarta-mvc/mvc/app/` gives us the response shown below.

[Hello world example] | *mvc/mvc-1.png*

Models

So far we have seen how we can pass models, or data to the view for display through the `Models` map.

Another way is through the use of CDI. First let's introduce our model, this time as a Plain Old Java Object (POJO), garnished with two CDI annotations, shown below.

```
@Named
@RequestScoped
public class Salutation {

    private String greet;
    private String platform;
    private String greetingDate;
    private String message;

    public String getGreet() {
        return greet;
    }

    public void setGreet(String greet) {
        this.greet = greet;
    }

    public String getPlatform() {
        return platform;
    }

    public void setPlatform(String platform) {
        this.platform = platform;
    }
```

```
    public String getGreetingDate() {
        return greetingDate;
    }

    public void setGreetingDate(String greetingDate) {
        this.greetingDate = greetingDate;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

}
```

Class `Salutation` is a simple Java class with some fields. These are the same fields we passed to the first view through the `Models` map.

`Salutation` is annotated `@Named` and `@RequestScoped`. `@Named` is a CDI qualifier that makes CDI managed instances of the class available in an Expression Language context - as used in the facelet files.

The `@RequestScoped` annotation will cause a new instance of `Salutation` to be created for each injection point.

With the model in place, let's look at the amended controller and how the model is instantiated and populated.

```
@Path("app")
@Controller
public class AppController {

    @Inject
    Salutation salutation;

    @GET
    @Path("salute")
    public String salute() {
        String formattedDate = LocalDateTime.now(ZoneOffset.UTC).format
(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));

        salutation.setGreet("Hello, World! Jakarta MVC");
        salutation.setPlatform("Jakarta EE 11 on Payara 7 Community");
        salutation.setGreetingDate(formattedDate);
        salutation.setMessage("Your Jakarta MVC application is running!");

        return "salute.xhtml";
```

```
    }

}
```

The `AppController` controller has a new method, `salute()`, hosted at the path `/salute`, that populates a CDI injected instance of class `Salutation`.

This method returns the `salute.xhtml` view to render the data. As you can see, the `Models` map is not used anywhere at all.

The injected `Salutation` instance is automatically available to the view thanks to the `@Named` annotation.

The `salute.xhtml` is shown next. This should be created under a `views` directory in the `WEB-INF` folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns:h="http://xmlns.jcp.org/jsf/html">

<h:head>
    <title>Title</title>
</h:head>

<h:body>
    <h1>#{salutation.greet}</h1>

    <p>#{salutation.message}</p>
    <p>This application is running on #{salutation.platform}, deployed on
#{salutation.greetingDate}</p>
</h:body>

</html>
```

The `salute.xhtml` uses the same `#{}` expression to access the model. This time around it calls the getter methods of the various fields.

The salutation instance is what is CDI makes available automatically. This way, the view has access to the model without explicitly using the `Models` map. The `salute` method is hosted at `http://localhost:8080/jakarta-mvc/mvc/app/salute`, which returns the following.

[Hello world example 2] | *mvc/mvc-2.png*

See Also

- Jakarta MVC Specification

Using Jakarta Enterprise Beans Technology

This chapter describes how Jakarta Enterprise Beans (EJB) technology is supported in Azul Payara.

For general information about enterprise beans, see "Enterprise Beans" in The Jakarta EE Tutorial.

> **NOTE**
>
> The Web Profile of the Payara Server supports the EJB 4.0+ Lite specification, which allows enterprise beans within web applications, among other features.
>
> The Full Profile supports the entire EJB 4.0+ specification. For details, see Jakarta Enterprise Beans.

Payara Server is backward compatible with 1.1, 2.0, 2.1, 3.0, 3.1 and 4.0 enterprise beans. However, to take advantage of version 3.1+ features, you should develop new beans as 3.1+ enterprise beans.

Customized EJB-JAR Archive Names

It is possible to instruct the server runtime to override the name of an EJB-JAR module when it is deployed either as a standalone module or as part of an EAR application. It is also possible to instruct Payara Server to override the module and/or application's name.

Overwriting the Module Name

When deploying an EJB-JAR module on Payara Server/Micro, the portable JNDI names for all scanned EJBs will be generated using the name of the module as specified on the *ejb-jar.xml* deployment descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns = "https://jakarta.ee/xml/ns/jakartaee"
         version = "4.0"
         xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation = "https://jakarta.ee/xml/ns/jakartaee
```

```
https://jakarta.ee/xml/ns/jakartaee/ejb-jar_4_0.xsd">
    <module-name>MODULE_NAME</module-name>
</ejb-jar>
```

NOTE	If the name's not specified in the deployment descriptor, the specification states that the module name will be the same as the JAR artifact used to deploy it.

However, when deploying a JAR from an IDE (like **NetBeans** or **IntelliJ**), the IDE deploys to Payara Server using the asadmin `deploy` command, with the `--name` option specified. This will force the module to have the specified name over the name defined in *ejb-jar.xml*. This is undesired because the IDE usually infers the module name from the name of the project or the JAR file and doesn't take the correct name of the module into account.

The module name defined in the deployment descriptor will be used even if it tries to be overridden using the `--name` option. This behaviour will **always** take precedence.

IMPORTANT	In the case you need to overwrite the name of the module when deploying the module, use the `--forceName` command option.

Overwriting the Application Name

In the case of EAR artifacts, the portable JNDI names for all scanned EJBs will use the application name defined in the *application.xml* deployment descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="https://jakarta.ee/xml/ns/jakartaee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/application_10.xsd" version="10">
    <display-name>My Application</display-name>
    <application-name>APPLICATION_NAME</application-name>
</application>
```

NOTE	You can use the `--name` and `--forcename` deployment options to override the application name in a similar manner with what happens with EJB-JAR modules.

Value Added Features

Azul Payara provides a number of value additions that relate to EJB development. References to more in-depth material are included.

Read-Only Beans

Another feature that Payara Server provides is the read-only bean, an EJB 2.1 entity bean that is never modified by an EJB client. Read-only beans avoid database updates completely.

IMPORTANT	Read-only beans are specific to Azul Payara and are not part of the Enterprise JavaBeans Specification, v2.1. Use of this feature for an EJB 2.1 bean results in a non-portable application. To make an EJB 3.0 entity read-only, use `@Column` annotations to mark its columns `insertable=false` and `updatable=false`.

A read-only bean can be used to cache a database entry that is frequently accessed but rarely updated (externally by other beans).

When the data that is cached by a read-only bean is updated by another bean, the read-only bean can be notified to refresh its cached data.

Payara Server provides a number of ways by which a read-only bean's state can be refreshed. By setting the `refresh-period-in-seconds` element in the `payara-ejb-jar.xml` file and the `trans-attribute` element (or `@TransactionAttribute` annotation) in the `ejb-jar.xml` file, it is easy to configure a read-only bean that is one of the following:

- Always refreshed
- Periodically refreshed
- Never refreshed
- Programmatically refreshed

NOTE	Read-only beans are best suited for situations where the underlying data

> never changes, or changes infrequently. For further information and usage guidelines, see Using Read-Only Beans.

The `pass-by-reference` Element

The `pass-by-reference` element in the `payara-ejb-jar.xml` file allows you to specify the parameter passing semantics for co-located remote EJB invocations. This is an opportunity to improve performance.

However, use of this feature results in non-portable applications. See "pass-by-reference" in the Payara Server Application Deployment section.

Pooling and Caching

The EJB container of Payara Server pools anonymous instances (message-driven beans, stateless session beans, and entity beans) to reduce the overhead of creating and destroying objects.

The EJB container maintains the free pool for each bean that is deployed. Bean instances in the free pool have no identity (that is, no primary key associated) and are used to serve method calls. The free beans are also used to serve all methods for stateless session beans.

Bean instances in the free pool transition from a **Pooled** state to a **Cached** state after `ejbCreate` and the business methods run. The size and behavior of each pool is controlled using pool-related properties in the EJB container or the `payara-ejb-jar.xml` file.

In addition, Payara Server supports a number of tunable parameters that can control the number of "stateful" instances (stateful session beans and entity beans) cached as well as the duration they are cached. Multiple bean instances that refer to the same database row in a table can be cached. The EJB container maintains a cache for each bean that is deployed.

To achieve scalability, the container selectively evicts some bean instances from the cache, usually when it overflows. These evicted bean instances return to the free bean

pool. The size and behavior of each cache can be controlled using the cache-related properties in the EJB container or the `payara-ejb-jar.xml` file.

Pooling and caching parameters for the `payara-ejb-jar.xml` file are described in "bean-cache" in the Payara Server Application Deployment section.

Pooling Parameters

One of the most important parameters for Payara Server pooling is `steady-pool-size`. When `steady-pool-size` is set to a value greater than `0`, the container not only pre-populates the bean pool with the specified number of beans, but also attempts to ensure that this number of beans is always available in the free pool. This ensures that there are enough beans in the ready-to-serve state to process user requests.

Note that the `steady-pool-size` and `max-pool-size` parameters only govern the number of instances that are pooled over a long period of time. They do not necessarily guarantee that the number of instances that may exist in the JVM at a given time will not exceed the value specified by `max-pool-size`.

For example, suppose an idle stateless session container has a fully-populated pool with a `steady-pool-size` of `10`. If `20` concurrent requests arrive for the EJB component, the container creates `10` additional instances to satisfy the burst of requests. The advantage of this is that it prevents the container from blocking any of the incoming requests. However, if the activity dies down to `10` or fewer concurrent requests, the additional `10` instances are discarded.

Another parameter, `pool-idle-timeout-in-seconds`, allows the administrator to specify the amount of time a bean instance can be idle in the pool. When `pool-idle-timeout-in-seconds` is set to greater than `0`, the container removes or destroys any bean instance that is idle for this specified duration.

Caching Parameters

Payara Server provides a way that completely avoids caching of entity beans, using commit option `C`. Commit option `C` is particularly useful if beans are accessed in large number but very rarely reused. For additional information, refer to commit-options.

Payara Server caches can be either bounded or unbounded. Bounded caches have limits on the number of beans that they can hold beyond which beans are passivated. For stateful session beans, there are three ways (LRU, NRU and FIFO) of picking victim beans when cache overflow occurs.

Caches can also passivate beans that are idle (not accessed for a specified duration).

Priority Based Scheduling of Remote Bean Invocations

You can create multiple thread pools, each having its own work queues. An optional element in the `payara-ejb-jar.xml` file, `use-thread-pool-id`, specifies the thread pool that processes the requests for the bean. The bean must have a remote interface, or `use-thread-pool-id` is ignored. You can create different thread pools and specify the appropriate thread pool ID for a bean that requires a quick response time.

NOTE	If there is no such thread pool configured or if the element is absent, the default thread pool is used.

Immediate Flushing

Normally, all entity bean updates within a transaction are batched and executed at the end of the transaction. The only exception is the database flush that precedes execution of a finder or select query.

Since a transaction often spans many method calls, you might want to find out if the updates made by a method succeeded or failed immediately after method execution. To force a flush at the end of a method's execution, use the `flush-at-end-of-method` element in the `payara-ejb-jar.xml` file. Only non-finder methods in an entity bean can be flush-enabled. (For an EJB 2.1 bean, these methods must be in the Local, Local Home, Remote, or Remote Home interface). See "flush-at-end-of-method" in the Payara Server Application Deployment section.

Upon completion of the method, the EJB container updates the database. Any exception thrown by the underlying data store is wrapped as follows:

- If the method that triggered the flush is a `create` method, the exception is wrapped with `CreateException`.
- If the method that triggered the flush is a `remove` method, the exception is wrapped with `RemoveException`.
- For all other methods, the exception is wrapped with `EJBException`.

All normal end-of-transaction database synchronization steps occur regardless of whether the database has been flushed during the transaction.

EJB Timer Service

The EJB Timer Service uses a database to store persistent information about EJB timers. The EJB Timer Service in Payara Server is preconfigured to use an embedded version of the H2 database via its `jdbc/__TimerPool` default datasource.

The EJB Timer Service configuration can store persistent timer information in any database supported by Payara Server for persistence. For configurations of supported drivers, see "configuration-specifics-for-jdbc-drivers" in the Payara Server General Administration section.

The timer service is automatically enabled when you deploy an application or module that uses it. You can verify that the timer service is running by accessing the following URL in a local context:

```
http://localhost:8080/ejb-timer-service-app/timer
```

To change the database used by the EJB Timer Service, set the EJB Timer Service's Timer DataSource setting to a valid JDBC resource. If the EJB Timer Service has already been started in a server instance, you must also create the timer database table. DDL files to create these tables are located in `as-install/lib/install/databases`.

Using the EJB Timer Service is equivalent to interacting with a single JDBC resource manager. If an EJB component or application accesses a database either directly through JDBC or indirectly (for example, through an entity bean's persistence

mechanism), and also interacts with the EJB Timer Service, its data source must be configured with an XA-compatible JDBC driver.

You can change the following EJB Timer Service settings. You must restart the server for these changes to take effect.

- **Minimum Delivery Interval**: Specifies the minimum time in milliseconds before an expiration for a particular timer can occur. This guards against tiny timer increments that can overload the server. The default value is `1000`.
- **Maximum Redeliveries**: Specifies the maximum number of times the EJB timer service attempts to redeliver a timer expiration after an exception or rollback of a container-managed transaction. The default value is `1`.
- **Redelivery Interval**: Specifies how long in milliseconds the EJB timer service waits after a failed `ejbTimeout` delivery before attempting a redelivery. The default is `5000`.
- **Timer DataSource**: Specifies the database used by the EJB Timer Service. The default is `jdbc/__TimerPool`.

CAUTION	Do not use the <code>jdbc/TimerPool</code> resource for timers in clustered Payara Server environments. You must instead use a custom JDBC resource or the <code>jdbc/default</code> resource. See the instructions below, in To Deploy an EJB Timer to a Cluster. Also refer to "enabling-the-jdbcdefault-resource-in-a-clustered-environment" in the Payara Server General Administration section.

For information about migrating EJB timers, see "migrating-ejb-timers" in the Payara Server High Availability section.

You can use the `--keepstate` option of the `asadmin redeploy` command to retain EJB timers between re-deployments.

The default value for `--keepstate` is `false`. This option is supported only on the default server instance (the DAS), named `server`. It is not supported and ignored for any other deployment target.

When the `--keepstate` is set to `true`, each application that uses an EJB timer is assigned an **ID** in the timer database. The EJB object that is associated with a given application is assigned an ID that is constructed from the application ID and a numerical suffix. To preserve active timer data, Payara Server stores the application ID and the EJB ID in the timer database. To restore the data, the class loader of the newly redeployed application retrieves the EJB timers that correspond to these IDs from the timer database.

Advanced Persistent Timer Configuration

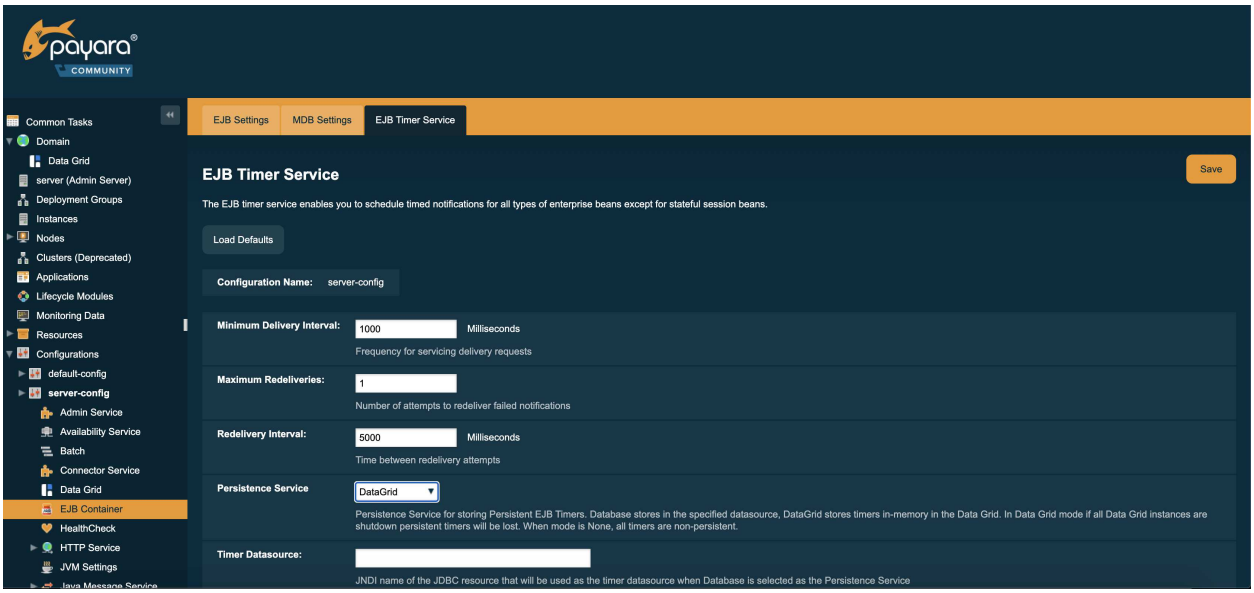
If using an external RDBMS engine for storing persistent timer data is not an option, it is also possible to use the **Domain Data Grid** to function as a replacement in production environments.

Persisting an EJB Timer to the Domain Data Grid means that the Data Grid itself will store the timer details, preserving it even if the original instance leaves the grid.

IMPORTANT	All stored timers are lost if the whole domain is stopped. In the case of Payara Micro, this will occur if all instances in the Data Grid are stopped as well.

The Persistence service for EJB Timers can be set in the Admin console by navigating to the **EJB Timer Service** tab in the EJB Container node of a configuration.

To use the Data Grid to store EJB Timers set the **Persistence Service** to *DataGrid*



EJB Timers will be coordinated across a single deployment group and if an instance of the deployment group is stopped another instance in the same deployment group will take ownership of the timer and ensure it is fired.

CAUTION	Clustered Timer Firing Mode is NOT used in this version of Payara Server and is always "One Per Deployment Group"

It is also possible to set the `ejb-timer-service` from the command line. To get the current state, run the following command:

```
asadmin> get configs.config.<your-config>.ejb-container.ejb-timer-service
```

This will return the current state taken from the *domain.xml* configuration file, which by default should be something similar to the following:

```
asadmin> get configs.config.server-config.ejb-container.ejb-timer-service
configs.config.server-config.ejb-container.ejb-timer-service.ejb-timer-service
=Database
configs.config.server-config.ejb-container.ejb-timer-service.max-redeliveries=1
configs.config.server-config.ejb-container.ejb-timer-service.minimum-delivery-
interval-in-millis=1000
configs.config.server-config.ejb-container.ejb-timer-service.redelivery-interval-
internal-in-millis=5000
Command get executed successfully.
```

To persist to the Data Grid you need only change the value for

`configs.config.server-config.ejb-container.ejb-timer-service.ejb-timer-service` to `DataGrid`.

To do this, run the following `set` command:

```
asadmin> set configs.config.server-config.ejb-container.ejb-timer-service.ejb-timer-service=DataGrid
```

WARNING | You will need to restart your domain to apply the changes.

Timer Migration

EJB Timers stored in the Domain Data Grid support timer migration between instances in the same Deployment Group. You can migrate timers using the Admin console from the **Deployment Group** page.

Timers can also be migrated between instances using asadmin commands like this:

```
asadmin> migrate-timers --target server1 server2
```

Where `server1` is the active instance to migrate timers to and `server2` is the failed instance.

Migration from Live Instances

The `migrate-timers` command can also be used to migrate timers that are scheduled to expire on a live instance. This allows a user to pre-emptively move their timers around without having to resort to failover mechanics.

To Deploy an EJB Timer to a Cluster

This procedure explains how to deploy an EJB timer to a cluster or deployment group.

By default, Payara Server's Timer Service points to the preconfigured `jdbc/__TimerPool` resource, which uses an embedded H2 database configuration that will not work in clustered environments.

The problem is that the embedded H2 database runs in the server's JVM, so when you

use the `jdbc/__TimerPool` resource, each DAS and each server instance will have their own, separate, databases.

Because of this, clustered server instances will not be able to find the database table on the DAS, and the DAS will not be able to find the tables on the clustered server instances.

The solution is to use either a custom JDBC resource or the `jdbc/__default` resource that is preconfigured but not enabled by default in Payara Server.

IMPORTANT	If you intend on creating a new timer data resource, it should be created BEFORE deploying applications that will use the timer.

Here are the steps on configuring the Timer Service:

1. Execute the following command:

```
asadmin set configs.config.cluster_name-config.ejb-container.ejb-timer-
service.timer-datasource=jdbc/my-timer-resource
```

2. Restart the DAS and the target cluster(s) or deployment groups.

```
asadmin stop-cluster cluster-name | deployment-group-name
asadmin stop-domain domain-name
asadmin start-domain domain-name
asadmin start-cluster cluster-name | deployment-group-name
```

Troubleshooting

If you inadvertently used the `jdbc/__TimerPool` resource for your EJB timer in a clustered Payara Server environment, the DAS and the server instances will be using separate H2 database instances that are running in individual JVMs.

For timers to work in a clustered environment, the DAS and the clustered server instances must share a common database.

If you attempt to deploy an application with EJB timers without setting the timer resource correctly, the startup will fail, and you will be left with a marker file, named

`ejb-timer-service-app`, on the DAS that will prevent the Timer Service from correctly creating the database table.

The solution is to remove the marker file on the DAS, restart the DAS and the clusters or deployment groups, and then redeploy any applications that rely on the offending EJB timer.

The marker file is located on the DAS in `domain-dir/generated/ejb/ejb-timer-service-app`.

Using Session Beans

This section provides guidelines for creating session beans in a Payara Server environment.

Information on session beans is contained in the Jakarta Enterprise Beans specification.

About the Session Bean Containers

Like an entity bean, a session bean can access a database through Java Database Connectivity (JDBC) calls. A session bean can also provide transaction settings. These transaction settings and JDBC calls are referenced by the session bean's container, allowing it to participate in transactions managed by the container.

A container managing stateless session beans has a different charter from a container managing stateful session beans.

Stateless Container

The stateless container manages stateless session beans, which, by definition, do not carry client-specific states. All session beans (of a particular type) are considered equal.

A stateless session bean container uses a bean pool to service requests. The Payara Server specific deployment descriptor file, `payara-ejb-jar.xml`, contains the properties that define the pool:

- `steady-pool-size`

- `resize-quantity`
- `max-pool-size`
- `pool-idle-timeout-in-seconds`

For more information about `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml`, see "the-payara-ejb-jar.xml-file" in the Payara Server Application Deployment section.

Payara Server provides the `wscompile` and `wsdeploy` tools to help you implement a web service endpoint as a stateless session bean.

It is possible to limit the number of concurrent Stateless EJB instances that are dispatched, allowing fine-grained control of resources, limiting surface area for DDOS attacks and making applications run more smoothly and efficiently.

This is done regardless of the maximum number of instances available in a bean's pool. Using these boundaries, it is possible to instruct the EJB container so that the maximum number of threads is not exceeded at runtime.

NOTE The system property `fish.payara.ejb-container.max-wait-time-in-millis` can be set to change the default global value of `<max-wait-time-in-millis>` for ***ALL*** Stateless EJB bean pools.

Example

The following is a sample `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` deployment descriptor that configures 2 EJBs with the settings mentioned beforehand.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payara-ejb-jar PUBLIC "-//Payara.fish//DTD Payara Application Server 7
EJB 4.0 Revision 1//EN"
"https://raw.githubusercontent.com/payara/Payara/refs/heads/main/appserver/deployme
nt/dtds/src/main/resources/glassfish/lib/dtds/payara7-ejb-jar_4_0-1.dtd">
<payara-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>PooledStatelessBean</ejb-name>
      <bean-pool>
          <max-pool-size>1</max-pool-size>
          <steady-pool-size>1</steady-pool-size>
      </bean-pool>
```

```
    </ejb>
    <ejb>
      <ejb-name>PooledMDB</ejb-name>
      <bean-pool>
          <max-pool-size>1</max-pool-size>
          <resize-quantity>1</resize-quantity>
      </bean-pool>
    </ejb>
  </enterprise-beans>
</payara-ejb-jar>
```

Stateful Container

The stateful container manages the stateful session beans, which, by definition, carry the client-specific state. There is a one-to-one relationship between the client and the stateful session beans. At creation, each stateful session bean (SFSB) is given a unique session ID that is used to access the session bean so that an instance of a stateful session bean is accessed by a single client only.

Stateful session beans are managed using an internal cache. The size and behavior of stateful session beans cache are controlled by specifying the following `payara-ejb-jar.xml` parameters:

- `max-cache-size`
- `resize-quantity`
- `cache-idle-timeout-in-seconds`
- `removal-timeout-in-seconds`
- `victim-selection-policy`

The `max-cache-size` element specifies the maximum number of session beans that are held in cache. If the cache overflows (when the number of beans exceeds `max-cache-size`), the container then passivates some beans or writes out the serialized state of the bean into a file. The directory in which the file is created is obtained from the EJB container using the configuration APIs.

For more information about `payara-ejb-jar.xml`, see "the-payara-ejb-jar.xml-file" in the Payara Server Application Deployment section.

The passivated beans are stored on the file system. The Session Store Location setting in the EJB container allows an administrator to specify the directory where passivated beans are stored. By default, passivated stateful session beans are stored in application-specific subdirectories created under `${domain-dir}/session-store`.

> **NOTE** Make sure the `delete` option is set in the `server.policy` file, or expired file-based sessions might not be deleted properly. For more information about `server.policy`, see the-server.policy-file.

The Session Store Location setting also determines where the session state is persisted if it is in a High Availability environment; see Choosing a Persistence Store.

Stateful Session Bean Failover

A stateful session bean's state can be saved in a persistent store in case a server instance fails. The state of an SFSB is saved to the persistent store at predefined points in its life cycle. This is called **checkpointing**. If SFSB checkpointing is enabled, checkpointing generally occurs after any transaction involving the SFSB is completed, even if the transaction rolls back.

However, if an SFSB participates in a bean-managed transaction, the transaction might be committed in the middle of the execution of a bean method. Since the bean's state might be undergoing transition as a result of the method invocation, this is not an appropriate instant to checkpoint the bean's state.

In this case, the EJB container checkpoints the bean's state at the end of the corresponding method, provided the bean is not in the scope of another transaction when that method ends. If a bean-managed transaction spans across multiple methods, checkpointing is delayed until there is no active transaction at the end of a subsequent method.

The state of an SFSB is not necessarily transactional and might be significantly modified as a result of non-transactional business methods. If this is the case for an SFSB, you can specify a list of checkpointed methods. If SFSB checkpointing is enabled,

checkpointing occurs after any checkpointed methods are completed.

The following table lists the types of references that SFSB failover supports. All objects bound into an SFSB must be one of the supported types. In the table, `No` indicates that failover for the object type might not work in all cases and that no failover support is provided. However, failover might work in some cases for that specific object type. For example, failover might work because the class implementing that type is **serializable**.

Java Object Type	Failover Support
Co-located or distributed stateless session, stateful session, or entity bean reference	Yes
JNDI context	Yes, `InitialContext` and `java:comp/env`
`UserTransaction`	Yes, but if the instance that fails is never restarted, any prepared global transactions are lost and might not be correctly rolled back or committed.
JDBC DataSource	No
Java Message Service (JMS) ConnectionFactory, Destination	No
JavaMail Session	No
Connection Factory	No
Administered Object	No
Web service reference	No
Serializable Java types	Yes
Extended persistence context	No

For more information about the `InitialContext`, see accessing-the-naming-context. For more information about transaction recovery, see using-the-transaction-service. For more information about Administered Objects, see "administering-jms-physical-destinations" in the Payara Server General Administration section.

If a server instance to which an RMI-IIOP client request is sent crashes during the

request processing (before the response is prepared and sent back to the client), an error is sent to the client.

The client must retry the request explicitly. When the client retries the request, the request is sent to another server instance in the cluster, which retrieves session state information for this client.

HTTP sessions can also be saved in a persistent store in case a server instance fails. In addition, if a distributable web application references an SFSB, and the web application's session fails over, the EJB reference is also failed over. For more information, see distributed-sessions-and-persistence.

If an SFSB that uses session persistence is un-deployed while the Payara Server instance is stopped, the session data in the persistence store might not be cleared. To prevent this, undeploy the SFSB while the Payara Server instance is running.

Configure SFSB failover by:

1. Choosing a Persistence Store
2. Enabling Checkpointing
3. Specifying Methods to Be Checkpointed

Choosing a Persistence Store

The following types of persistent storage are supported for passivation and checkpointing of the SFSB state:

The local file system: Allows a single server instance to recover the SFSB state after a failure and restart. This store also provides passivation and activation of the state to help control the amount of memory used. This option is not supported in a production environment that requires SFSB state persistence.

+ NOTE: This is the default storage mechanism if availability is not enabled.

Other servers: Uses other server instances in the cluster for session persistence.

Clustered server instances replicate session state. Each backup instance stores the replicated data in memory. This is the default storage mechanism if availability is enabled.

Choose the persistence store in one of the following ways:

- To use the local file system, first disable availability. Select the *Availability Service* component under the relevant configuration in the Administration Console. Uncheck the *Availability Service* box. Then select the EJB Container component and edit the Session Store Location value. The default is `${domain-dir}/session-store`.
- To use other servers, select the *Availability Service* component under the relevant configuration in the Administration Console. Check the *Availability Service* box. To enable availability for the EJB container, select the EJB Container Availability tab, then check the Availability Service box.

All instances in a Payara Server cluster or deployment group must have the same availability settings to ensure consistent behavior.

Using the `--keepstate` Option

If you are using the file system for persistence, you can use the `--keepstate` option of the `asadmin redeploy` command to retain the SFSB state between re-deployments.

The default value for `--keepstate` is `false`. This option is supported only on the default server instance (the DAS), named `server`. It is not supported and ignored for any other deployment target.

Some changes to an application between re-deployments prevent this feature from working properly. For example, do not change the set of instance variables in the SFSB bean class.

If any active SFSB instance fails to be preserved or restored, none of the SFSB instances will be available when the redeployment is complete. However, the redeployment continues and a warning is logged.

To preserve active state data, Payara Server serializes the data and saves it in memory. To restore the data, the class loader of the newly redeployed application deserializes the data that was previously saved.

Using the `--asyncreplication` Option

If you are using replication on other servers for persistence, you can use the `--asyncreplication` option of the `asadmin deploy` command to specify that SFSB states are first buffered and then replicated using a separate asynchronous thread.

If `--asyncreplication` is set to `true` (the default), performance is improved but availability is reduced. If the instance where states are buffered but not yet replicated fails, the states will be lost.

If set to `false`, performance is reduced but availability is guaranteed. States are not buffered but immediately transmitted to other instances in the cluster or deployment group.

Enabling Checkpointing

Server Instance and EJB Container Levels

To enable SFSB checkpointing at the server instance or EJB container level, see Choosing a Persistence Store.

Application and EJB Module Levels

To enable SFSB checkpointing at the application or EJB module level during deployment, use the `asadmin deploy` command with the `--availabilityenabled` option set to `true`.

SFSB Level

To enable SFSB checkpointing at the bean level, set `availability-enabled="true"` in the `ejb` element of the bean definition in the `payara-ejb-jar.xml` file as follows:

```
<payara-ejb-jar>
```

```
    ...
    <enterprise-beans>
       ...
       <ejb availability-enabled="true">
          <ejb-name>MySFSB</ejb-name>
       </ejb>
    ...
    </enterprise-beans>
</payara-ejb-jar>
```

Specifying Methods to Be Checkpointed

If SFSB checkpointing is enabled, checkpointing generally occurs after any transaction involving the SFSB is completed, even if the transaction rolls back.

To specify additional optional checkpointing of SFSBs at the end of non-transactional business methods that cause important modifications to the bean's state, use the `checkpoint-at-end-of-method` element within the `ejb` element in `payara-ejb-jar.xml`.

For example:

```
<payara-ejb-jar>
    ...
    <enterprise-beans>
       ...
       <ejb availability-enabled="true">
          <ejb-name>ShoppingCartEJB</ejb-name>
          <checkpoint-at-end-of-method>
             <method>
                <method-name>addToCart</method-name>
             </method>
          </checkpoint-at-end-of-method>
       </ejb>
       ...
    </enterprise-beans>
</payara-ejb-jar>
```

For details, see "checkpoint-at-end-of-method" in the Payara Server Application Deployment section.

The non-transactional methods in the `checkpoint-at-end-of-method` element can be the following:

- `create` methods defined in the home or business interface of the SFSB, if you want

to checkpoint the initial state of the SFSB immediately after creation.

- For SFSBs using container managed transactions only, methods in the remote interface of the bean marked with the transaction attribute `TX_NOT_SUPPORTED` or `TX_NEVER`.
- For SFSBs using bean managed transactions only, methods in which a bean managed transaction is neither started nor committed.

Any other methods mentioned in this list are ignored. At the end of invocation of each of these methods, the EJB container saves the state of the SFSB to the persistent store.

NOTE

If an SFSB does not participate in any transaction, and if none of its methods are explicitly specified in the `checkpoint-at-end-of-method` element, the bean's state is not checkpointed at all even if `availability-enabled="true"` for this bean.

For better performance, specify a small subset of methods. The methods chosen should accomplish a significant amount of work in the context of the Jakarta EE application or should result in some important modification to the bean's state.

Session Bean Restrictions and Optimizations

This section discusses restrictions on developing session beans and provides some optimization guidelines.

Optimizing Session Bean Performance

For stateful session beans, co-locating the stateful beans with their clients so that the client and bean are executing in the same process address space improves performance.

Restricting Transactions

The following restrictions on transactions are enforced by the container and must be observed as session beans are developed:

- A session bean can participate in, at most, a single transaction at a time.

- If a session bean is participating in a transaction, a client cannot invoke a method on the bean such that the `trans-attribute` element (or `@TransactionAttribute` annotation) in the `ejb-jar.xml` file would cause the container to execute the method in a different or unspecified transaction context or an exception is thrown.
- If a session bean instance is participating in a transaction, a client cannot invoke the `remove` method on the session object's home or business interface object, or an exception is thrown.

EJB Singletons

EJB Singletons are created for each server instance in a cluster, and not once per cluster.

Using Read-Only Beans

A read-only bean is an EJB 2.1 entity bean that is never modified by an EJB client. The data that a read-only bean represents can be updated externally by other enterprise beans, or by other means, such as direct database updates.

NOTE Read-only beans are specific to Payara Server and are not part of the Jakarta Enterprise Beans Specification. Use of this feature for an EJB 2.1 bean results in a non-portable application.

TIP To make an EJB 3.0+ entity bean read-only, use `@Column` annotations to mark its columns `insertable=false` and `updatable=false`.

Read-only beans are best suited for situations where the underlying data never changes, or changes infrequently.

Read-Only Bean Characteristics and Life Cycle

Read-only beans are best suited for situations where the underlying data never changes, or changes infrequently. For example, a read-only bean can be used to represent a stock quote for a particular company, which is updated externally. In such a case, using a regular entity bean might incur the burden of calling `ejbStore`, which can be avoided by using a read-only bean.

Read-only beans have the following characteristics:

- Only entity beans can be read-only beans.
- Only container-managed transactions are allowed; read-only beans cannot start their own transactions.
- Read-only beans don't update any bean state.
- `ejbStore` is never called by the container.
- `ejbLoad` is called only when a transactional method is called or when the bean is initially created (in the cache), or at regular intervals controlled by the bean's `refresh-period-in-seconds` element in the `payara-ejb-jar.xml` file.
- The home interface can have any number of find methods. The return type of the find methods must be the primary key for the same bean type (or a collection of primary keys).
- If the data that the bean represents can change, then `refresh-period-in-seconds` must be set to refresh the beans at regular intervals. `ejbLoad` is called at this regular interval.

A read-only bean comes into existence using the appropriate find methods.

Read-only beans are cached and have the same cache properties as entity beans. When a read-only bean is selected as a "victim" to make room in the cache, `ejbPassivate` is called and the bean is returned to the free pool. When in the free pool, the bean has no identity and is used only to serve any finder requests.

Read-only beans are bound to the naming service like regular read-write entity beans, and clients can look up read-only beans the same way read-write entity beans are looked up.

Read-Only Bean Good Practices

For best results, follow these guidelines when developing read-only beans:

- Avoid having any `create` or `remove` methods in the home interface.

- Use any of the valid EJB 2.1 transaction attributes for the `trans-attribute` element.

  The reason for having `TX_SUPPORTED` is to allow reading uncommitted data in the same transaction. Also, the transaction attributes can be used to force `ejbLoad`.

Refreshing Read-Only Beans

There are several ways of refreshing read-only beans, as addressed in the following sections:

Invoking a Transactional Method

Invoking any transactional method invokes `ejbLoad`.

Refreshing Periodically

Use the `refresh-period-in-seconds` element in the `payara-ejb-jar.xml` file to refresh a read-only bean periodically.

- If the value specified in `refresh-period-in-seconds` is zero or not specified, which is the default, the bean is never refreshed (unless a transactional method is accessed).
- If the value is greater than zero, the bean is refreshed at the rate specified.

NOTE	This is the only way to refresh the bean state if the data can be modified external to Payara Server.

By default, a single timer is used for all instances of a read-only bean. When that timer fires, all bean instances are marked as expired and are refreshed from the database the next time they are used.

Use the `-Dcom.sun.ejb.containers.readonly.relative.refresh.mode=true` flag to refresh each bean instance independently upon access if its refresh period has expired. Its default value is `false`. Note that each instance still has the same refresh period.This additional level of granularity can improve the performance of read-only

beans that do not need to be refreshed at the same time.

To set this flag, use the `asadmin create-jvm-options` command. For example:

```
asadmin create-jvm-options -Dcom.sun.ejb.containers.readonly.relative.refresh.mode
=true
```

Deploying Read-Only Beans

Read-only beans are deployed in the same manner as other entity beans. However, in the entry for the bean in the `payara-ejb-jar.xml` file, the `is-read-only-bean` element must be set to `true`. That is:

```
<is-read-only-bean>true</is-read-only-bean>
```

Also, the `refresh-period-in-seconds` element in the `payara-ejb-jar.xml` file can be set to some value that specifies the rate at which the bean is refreshed. If this element is missing, no refresh occurs.

All requests in the same transaction context are routed to the same read-only bean instance. Set the `allow-concurrent-access` element to either `true` (to allow concurrent accesses) or `false` (to serialize concurrent access to the same read-only bean). The default is `false`.

For further information on these elements, refer to "the-payara-ejb-jar.xml-file" in the Payara Server Application Deployment section.

Using Message-Driven Beans

This section describes message-driven beans and explains the requirements for creating them in a Payara Server environment.

Message-Driven Bean Configuration

For information about setting up load balancing for message-driven beans, see load-balanced-message-inflow.

Connection Factory and Destination

A message-driven bean is a client to a Connector inbound resource adapter. The message-driven bean container uses the JMS service integrated into Payara Server for message-driven beans that are JMS clients. JMS clients use JMS Connection Factory- and Destination-administered objects. A JMS Connection Factory administered object is a resource manager Connection Factory object that is used to create connections to the JMS provider.

The `mdb-connection-factory` element in the `payara-ejb-jar.xml` file for a message-driven bean specifies the connection factory that creates the container connection to the JMS provider.

The `jndi-name` element of the `ejb` element in the `payara-ejb-jar.xml` file specifies the JNDI name of the administered object for the JMS Queue or Topic destination that is associated with the message-driven bean.

Message-Driven Bean Pool

The container manages a pool of message-driven beans for the concurrent processing of a stream of messages. The `payara-ejb-jar.xml` file contains the elements that define the pool (that is, the `bean-pool` element):

- `steady-pool-size`
- `resize-quantity`
- `max-pool-size`
- `pool-idle-timeout-in-seconds`

For more information about `payara-ejb-jar.xml`, see "the-payara-ejb-jar.xml-file" in the Payara Server Application Deployment section.

Domain-Level Settings

You can control the following domain-level message-driven bean settings in the EJB container:

- **Initial and Minimum Pool Size**: Specifies the initial and minimum number of beans maintained in the pool. The default is `0`.
- **Maximum Pool Size**: Specifies the maximum number of beans that can be created to satisfy client requests. The default is `32`.
- **Pool Resize Quantity**: Specifies the number of beans to be created if a request arrives when the pool is empty (subject to the **Initial** and **Minimum Pool Size**), or the number of beans to remove if idle for more than the Idle Timeout. The default is `8`.
- **Idle Timeout**: Specifies the maximum time in seconds that a bean can remain idle in the pool. After this amount of time, the bean is destroyed. The default is `600` (10 minutes). A value of `0` means a bean can remain idle indefinitely.

Select the *Instances* component, select the instance from the table, and select the Monitor tab.

NOTE	Running monitoring when it is not needed might impact performance, so you might choose to turn monitoring off when it is not in use. For details, see "administering-the-monitoring-service" in the Payara Server General Administration section.

Message-Driven Bean Restrictions and Optimizations

This section discusses the following restrictions and performance optimizations that pertain to developing message-driven beans:

Pool Tuning and Monitoring

The message-driven bean pool is also a pool of threads, with each message-driven bean instance in the pool associating with a server session, and each server session associating with a thread.

Therefore, a large pool size also means a high number of threads, which impacts performance and server resources.

When configuring message-driven bean pool properties, make sure to consider factors such as message arrival rate and pattern, `onMessage` method processing time, overall

server resources (threads, memory, and so on), and any concurrency requirements and limitations from other resources that the message-driven bean accesses.

When tuning performance and resource usage, make sure to consider potential JMS provider properties for the connection factory used by the container (the `mdb-connection-factory` element in the `payara-ejb-jar.xml` file). For example, you can tune the Open Message Queue flow control related properties for connection factory in situations where the message incoming rate is much higher than `max-pool-size` can handle.

Refer to "administering-the-monitoring-service" in the Payara Server General Administration section for information on how to get message-driven bean pool statistics.

Fine-Tuning with the Activationconfigproperty Annotation

It is also possible to configure the MDB bean pool size with the `ActivationConfigProperty` annotation. Any bean annotated with `@MessageDriven` can use `@ActivationConfigProperty` to set property names and property values. For example:

```
@ActivationConfigProperty(propertyName = "MaxPoolSize", propertyValue = "100")
public class CustomerListener implements MessageListener{
}
```

The `MaxPoolSize`, `MaxWaitTimeInMillis`, `PoolResizeQuantity`, `SteadyPoolSize` and `PoolIdleTimeoutInSeconds` are all MDB pool properties that can configured using the `@ActivationConfigProperty` annotation.

The `onMessage` Runtime Exception

Message-driven beans, like other well-behaved `MessageListener` implementations, should not, in general, throw runtime exceptions. If a message-driven bean's `onMessage` method encounters a system-level exception or error that does not allow the method to successfully complete, the Jakarta Enterprise Beans Specification provides the following guidelines:

- If the bean method encounters a system exception or error, it should simply propagate the error from the bean method to the container (i.e., the bean method does not have to catch the exception).
- If the bean method performs an operation that results in a checked exception that the bean method cannot recover, the bean method should throw the `jakarta.ejb.EJBException` that wraps the original exception.
- Any other unexpected error conditions should be reported using the `jakarta.ejb.EJBException`.

Under container-managed transaction demarcation, upon receiving a runtime exception from a message-driven bean's `onMessage` method, the container rolls back the container-started transaction and the message is redelivered. This is because the message delivery itself is part of the container-started transaction.

By default, the Payara Server container closes the container's connection to the JMS provider when the first runtime exception is received from a message-driven bean instance's `onMessage` method.

This avoids potential message redelivery looping and protects server resources if the message-driven bean's `onMessage` method continues misbehaving. To change this default container behavior, use the `cmt-max-runtime-exceptions` property of the MDB container.

Here is an example `asadmin set` command that sets this property:

```
asadmin set server-config.mdb-container.property.cmt-max-runtime-exceptions="5"
```

The `cmt-max-runtime-exceptions` property specifies the maximum number of runtime exceptions allowed from a message-driven bean's `onMessage` method before the container starts to close the container's connection to the message source. By default, this value is set to `1`; `-1` disables this container protection.

A message-driven bean's `onMessage` method can use the

`jakarta.jms.Message.getJMSRedelivered` method to check whether a received message is a redelivered message.

CAUTION	The `cmt-max-runtime-exceptions` property is deprecated and support for this feature may be removed in the future.

EJB Timers in Deployment Groups

NOTE	This section only applies to Payara Server environments.

Timer facilities provided by the EJB APIs `TimerService` and `@Scheduled` need special configuration in order to work intuitively for applications targeting a deployment group. It is important to understand how EJB timers run (or expire) when configured on deployment group scenarios.

Before stating the specifics, keep in mind that there are four types of timers distinguished by their lifecycle (**persistent**, **non-persistent**) and the means of their definition (**declarative**, **programmatic**).

- Declarative timers are defined in EJB deployment descriptors or with the `@Scheduled` annotation.
- Programmatic timers are created via calls to the `TimerService.createTimer` method and its equivalents.

Non-Persistent Timers

Non-Persistent timers exist in memory only for the duration in which the application runs on the instances that belong to a deployment group.

Declarative non-persistent timers run/expire on every instance, whereas **programmatic ones only run/expire on the instance that created them**. When the instance shuts down, the timer will stop its expiration.

Standalone instances do not see each other's non-persistent timers, so calling `TimerService.getTimers()` only returns local non-persistent timers.

Non-Persistent timers have a simpler lifecycle but are not the default option. To explicitly define a non-persistent timer, the `@Schedule` annotation needs the `persistent=false` attribute, or the `TimerConfig` objects need to have their property `persistent` set to `false` explicitly.

Persistent Timers

Persistent timers exist in the persistent store for the entire time for which application is deployed to its corresponding domain.

Contrary to non-persistent ones, each persistent timer is **only executed on a single instance of a deployment group**. When the instance shuts down, each persistent timer migrates to another instance in the application's deployment group and continues firing there.

Since persistent timers are shared, the `TimerService.getTimers()` method returns all persistent timers in the deployment group, regardless of which instance created them.

TIP	EJB Timers are persistent by default.

Preconditions for Using Persistent Timers in Deployment Groups

Because "deployment groups" do not need to share configurations amongst their instances, there are several configuration and operational constraints that are not enforced by the server.

To ensure correct behaviour of persistent timers within a deployment group, the following conditions need to be met:

- The EJB Timer Service needs to use shared storage --an external database-- or "the DataGrid".
- All instances of a deployment group, and the Domain Administration Server MUST share the same configuration settings for the EJB Timer Service (see ejb-timer-service for more information).

- Applications with persistent timers are deployed to a deployment group and not managed on instance level afterward.
- Instances are members of only a single deployment group

EJB Timers may behave in unexpected ways if these conditions are not met.

Tracing Remote Ejbs

Remote calls to an EJB from a Java SE client will have their active OpenTelemetry span context automatically propagated to the server, with the server-side span being created as a child of this client call. This will allow users to trace EJB calls that are originated externally from the server, and the span context will be propagated internally to any traceable components.

Java SE Client Tracing

No additional setup is required on the client side over what is to be expected for making regular un-traced remote EJB calls - the requirements are the same: usage of `payara-embedded-all` or the application client facilities of a Payara Server instance.

No additional properties need to be specified when performing the initial context lookup either:

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.util.Properties;

public class MainClass{
    public static void main(String[] args){
        var contextProperties = new Properties();
        contextProperties.setProperty(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.enterprise.naming.SerialInitContextFactory");

        try {
            Context context = new InitialContext(contextProperties);
            EjbRemote ejb = (EjbRemote) context.lookup
("java:global/myRemoteEjb/Ejb");
        } catch (NamingException ne) {
            logger.warning("Failed performing lookup:\n" + ne.getMessage());
        }
    }
}
```

Getting a Tracer Instance

Injection of an OpenTracing tracer is not supported on Java SE clients, so you must create an instance yourself and register it to a `GlobalTracer` instance.

WARNING	Manual registration of a tracer instance is only required if using a third-party tracer such as ZipKin or Jaeger - if using the built-in Azul Payara Request Tracing service the registration will automatically happen during creation of the initial context.

NOTE	The built-in Payara Request Tracing service does not support tracing of Java SE clients (though will still propagate the active span context to the server)

```
import io.opentracing.Tracer;
import io.opentracing.util.GlobalTracer;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.util.Properties;

public class MainClass{

    public static void main(String[] args){
        var contextProperties = new Properties();
        contextProperties.setProperty(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.enterprise.naming.SerialInitContextFactory");

        try {
            Context context = new InitialContext(contextProperties);
            EjbRemote ejb = (EjbRemote) context.lookup
("java:global/myRemoteEjb/Ejb");

            Tracer tracer = GlobalTracer.get();
        } catch (NamingException ne) {
            logger.warning("Failed performing lookup:\n" + ne.getMessage());
        }
    }
}
```

Extra information on creating a Tracer instance and registering it to the `GlobalTracer` can be found here, but in short it can be done like so (replacing `CustomTracer` with your desired implementation):

```
public class MainClass{

    public static void main(String[] args){
        Tracer tracerImpl = new CustomTracer();
        GlobalTracer.register(tracerImpl);
    }
}
```

Starting a Span

The `@Traced` annotation is not supported on Java SE client methods, so spans must be started and finished manually. Note that the Span Context of the **active** span will be propagated to the server, so it is recommended that you use a try-with-resources clause to help ensure your span scope is the one you expect.

```
import io.opentracing.Scope;
import io.opentracing.Span;
import io.opentracing.Tracer;
import io.opentracing.util.GlobalTracer;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.util.Properties;

public class TracerClass{

    public static void main(String[] args){
        var contextProperties = new Properties();
        contextProperties.setProperty(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.enterprise.naming.SerialInitContextFactory");

        try {
            Context context = new InitialContext(contextProperties);
            EjbRemote ejb = (EjbRemote) context.lookup
("java:global/myRemoteEjb/Ejb");

            Tracer tracer = GlobalTracer.get();

            try (Scope scope = tracer.buildSpan("ExecuteEjb").startActive(true)) {
                ejb.doTheThing();
            }
        } catch (NamingException ne) {
            logger.warning("Failed performing lookup:\n" + ne.getMessage());
        }
    }
}
```

Once your span has been started, you can attach any desired baggage items (String key:value pairs), and these will be propagated to the server along with the span context.

These can then be retrieved within the EJB implementation hosted on Payara Server like so:

```
import io.opentracing.Span;
import io.opentracing.Tracer;

import jakarta.ejb.Stateless;
import jakarta.inject.Inject;

@Stateless
public class Ejb implements EjbRemote {

    @Inject
    Tracer tracer;

    public void businessMethod(){
        Span activeSpan = tracer.activeSpan();
        if (activeSpan != null) {
            String myBaggageItem = activeSpan.getBaggageItem("myBaggageItem");
        }
    }
}
```

NOTE	Baggage items are attached to the Span you invoke `setBaggageItem` on, as well as any child spans, so you are not prevented from starting additional child spans on your EJB methods manually or via the `@Traced` annotation.

JDK 17+ Support Considerations

The following considerations must be taken into account when using EJB Features on JDK 17+:

- If using a Java SE client for remote EJB lookups, add the `--add-opens=java.base/java.lang=ALL-UNNAMED` JVM argument to the list of startup arguments of the client.

Using Lite Remote EJB Technology

For scenarios when communication with remote EJBs is needed across a complex network topology or when a server architecture is deployed on a cloud provider, there's a new feature for the EJB server and client applications: A lite and thin client that can be used to communicate via HTTP(S) instead of IIOP-RMI.

Payara Server has support for classic remote EJB since its inception. Classic remote EJB uses the **IIOP-RMI** (Internet Inter-ORB Protocol Remote Method Invocation) protocol for transport communication and the CSIv2 (Common Secure Interoperability Protocol Version 2) protocol for security.

While feature rich, these protocols were not designed with firewalls, NAT, (private) clouds, Docker and generally the Internet in mind. For these environments the requirements of the protocol, such as establishing independent connections back from the server to the client, are too troublesome.

The **IIOP-RMI/CSIv2** also requires the use of multiple heavyweight client libraries. In the case of Payara Server the client library (either referenced by the `gf-client.jar` library or the `payara-embedded-all` dependency) is almost as big as the entire server as it's essentially a special form of the ACC (Application Client Container). For this reason a much-needed modernization of the transport and security protocols was needed, and for the environments described only one protocol really works, and that is the HTTP(S) protocol.

Payara Server features an additional complimentary EJB remoting technology based on the HTTP(S) protocol while favoring the use of a thin client library that make remote applications much lighter while circumventing known challenges for the Internet and other scenarios.

IMPORTANT	HTTP(S) for EJB Remoting in Payara Server is a completely different feature from the classic IIOP-RMI/CSIv2-based feature, and does not intend to replace the transport layer protocol used on it, rather it is implemented as an additional independent feature.
IMPORTANT	TX Propagation and Security Propagation is **not supported** by EJB Remoting in Payara Server.
WARNING	In the current version of this feature, not all remote EJB features are

implemented yet.

Configuring HTTP(S) for Remote Ejbs

HTTP(S) for EJB remoting in Payara Server makes use of a special web endpoint on which is located by default on the path `/ejb-invoker`.

IMPORTANT | This endpoint is disabled out of the box, and it can be enabled using the `set-ejb-invoker-configuration` asadmin command.

Configuring the Thin Client Dependency

Client applications that wish to use HTTP(S) as the transport protocol when calling remote EJBs will have to use a **special thin-client** dependency. To do this, you can add the following Maven dependency to your client project:

```
<dependency>
    <groupId>fish.payara.extras</groupId>
    <artifactId>ejb-http-client</artifactId>
    <version>{release-id}</version>
</dependency>
```

This artifact requires a patched version of the Jersey client which is pulled intransitive. This patched version is available within our Payara Nexus Repository. The Maven repository definition is present in the pom.xml file of the `ejb-http-client` artifact. However, when your dependencies are managed by a private artifact repository, for example, you might need to update it to refer to the Payara Nexus repository.

Also, we recommend having a look at the BOM feature so that versions can be defined more consistently.

Finally, the code that executes the call to the remote EJB must be modified in some manner. To obtain a type-safe proxy for any remote EJB bean, the traditional approach via JNDI is still used. An example is given below:

1. First, consider the following remote EJB interface:

```
@Remote
public interface BeanRemote {
    String method();
}
```

2. Second, consider a (secured) EJB that implements that interface and resides in a EJB application called "test" deployed on a Payara server instance that is listening in `http://localhost:8080`:

```
@Stateless
public class Bean implements BeanRemote, Serializable {
    private static final long serialVersionUID = 1L;
    @Override
    @RolesAllowed("g1")
    public String method() {
        return "method";
    }
}
```

3. Given the above, the following client code can be used to obtain a proxy to the `BeanRemote` bean and invoke a remote method defined on it:

```
import static javax.naming.Context.INITIAL_CONTEXT_FACTORY;
import static javax.naming.Context.PROVIDER_URL;
import java.util.Properties;
import javax.naming.Context;
import javax.naming.InitialContext;

public class RemoteEJBLiteClient{
    public static void main(String... args) throws NamingException{
        Properties environment = new Properties();
        environment.put(INITIAL_CONTEXT_FACTORY,
"fish.payara.ejb.http.client.RemoteEJBContextFactory");
        environment.put(PROVIDER_URL, "http://localhost:8080/ejb-invoker");
        InitialContext ejbRemoteContext = new InitialContext(environment);
        BeanRemote beanRemote = (BeanRemote) ejbRemoteContext.lookup
("java:global/test/Bean");
        beanRemote.method(); // returns "method"
    }
}
```

Calling a Secured Remote EJB

When calling a secured EJB using the `ejb-invoker` endpoint, there are some considerations in place for the client code:

1. If the remote EJB is secured with transport confidentiality (and integrity) enabled like

this:

```
<ejb>
    <ejb-name>Bean</ejb-name>
    <ior-security-config>
        <transport-config>
            <integrity>REQUIRED</integrity>
            <confidentiality>REQUIRED</confidentiality>
            <establish-trust-in-target>SUPPORTED</establish-trust-in-target>
            <establish-trust-in-client>SUPPORTED</establish-trust-in-client>
        </transport-config>
    </ior-security-config>
</ejb>
```

Then, the corresponding HTTP endpoint to use would be `https://localhost:8181/ejb-invoker` instead.

2. If the remote EJB also has authentication enabled (via username and password credentials) like this:

```
<ejb>
    <ejb-name>Bean</ejb-name>
    <as-context>
        <auth-method>USERNAME_PASSWORD</auth-method>
        <realm>default</realm>
        <required>true</required>
    </as-context>
</ejb>
```

Then the credentials required to correctly authenticate the user for the call have to be specified in the JNDI context with the following properties:

   - `javax.naming.Context.SECURITY_PRINCIPAL` for the username
   - `javax.naming.Context.SECURITY_CREDENTIALS` for the password

Here's an example of the complete client code used to call a secured remote EJB:

```
import static javax.naming.Context.INITIAL_CONTEXT_FACTORY;
import static javax.naming.Context.PROVIDER_URL;
import static javax.naming.Context.SECURITY_CREDENTIALS;
import static javax.naming.Context.SECURITY_PRINCIPAL;
import java.util.Properties;
import javax.naming.Context;
import javax.naming.InitialContext;
public class RemoteEJBLiteClient{
```

```
    public static void main(String... args) throws NamingException{
        Properties environment = new Properties();
        environment.put(INITIAL_CONTEXT_FACTORY
,"fish.payara.ejb.http.client.RemoteEJBContextFactory");
        environment.put(PROVIDER_URL, "https://localhost:8181/ejb-invoker");
        environment.put(SECURITY_PRINCIPAL, "u1");
        environment.put(SECURITY_CREDENTIALS, "p1");
        InitialContext ejbRemoteContext = new InitialContext(environment);
        BeanRemote beanRemote = (BeanRemote) ejbRemoteContext.lookup
("java:global/test/Bean");
        beanRemote.method(); // returns "method"
    }
}
```

IMPORTANT	When accessing secured EJBs you **should** use only HTTPS (that means, enabling confidential transport requirements to the remote EJB), as the submitted credentials will be transferred in clear text (not encrypted, only base64 encoded), which is a security risk you should avoid in any production environment.

JNDI Customization Options

Under the covers the remote EJB proxy uses a JAX-RS (Jersey) REST client builder in order to establish communication with the remote server. If you want to customize and modify the parameters for this communication (timeouts, keystores, etc.) the following JNDI context properties can be used to this end:

Table 192. `JNDI Options for Custom HTTP(S) Communication`

Property	Behaviour	Type
`fish.payara.connectTimeout`	The connection timeout. A value of **0** represents that the wait is indefinite. Negative values are not allowed. Unit is microseconds.	`Number` (from which it's `Long` value is taken) or a `String` that can be converted to a `Long` value.

Property	Behaviour	Type
`fish.payara.readTimeout`	The timeout to read a response. If the remote Payara doesn't respond within the defined time a ProcessingException is thrown with a `TimeoutException` as its cause. A value of **0** represents that the wait is indefinite. Negative values are not allowed. Unit is microseconds.	`Number` (from which it's `Long` value is taken) or a `String` that can be converted to a `Long` value.
`fish.payara.keyStore`	The key store to be used by the proxy. The key store contains the private key as well as certificates with its associated public keys.	Instance of `java.security.KeyStore` or a `String` representing its fully qualified classname.
`fish.payara.trustStore`	The trust store to be used by the proxy. The trust store must contain the certificates that are needed to communicate with the remote Payara Server.	Instance of `java.security.KeyStore` or a `String` representing its fully qualified classname.
`fish.payara.sslContext`	The SSL context that will be used by the proxy for creating secured connections to the Payara remote server. This context **must** be fully initialized, including the trust and key managers. Should not be used in conjunction with the `fish.payara.keyStore` and/or `fish.payara.trustStore` properties.	Instance of `javax.net.ssl.SSLContext` or a `String` representing its fully qualified classname.
`fish.payara.hostnameVerifier`	The hostname verifier to be used by the proxy to verify the endpoint's hostname against the identification information of it.	Instance of a `javax.net.ssl.HostnameVerifier` or a `String` representing its fully qualified classname.
`fish.payara.provider.principal`	The principal to be used by the proxy to access the secured endpoint.	Instance of a `String`.

Property	Behaviour	Type
`fish.payara.provider.credentials`	The credentials to be used by the proxy to access the secured endpoint.	Instance of a `String`.
`fish.payara.provider.authType`	To specify the authentication type of the secured endpoint. By default `BASIC` is defined.	Instance of a `String`.
`fish.payara.requestFilter`	To register the custom filter for the client request invoked by proxy.	Instance of a `jakarta.ws.rs.client.ClientRequestFilter` or a `Class` implementing `jakarta.ws.rs.client.ClientRequestFilter`.
`fish.payara.responseFilter`	To register the custom filter for the client response received by proxy.	Instance of a `jakarta.ws.rs.client.ClientResponseFilter ` or a `Class` implementing `jakarta.ws.rs.client.ClientResponseFilter`.
`fish.payara.executorService`	The executor service that will be used for executing asynchronous tasks. *(for future use)*	Instance of `java.util.concurrent.ExecutorService` or a `String` representing its fully qualified classname.
`fish.payara.scheduledExecutorService`	The executor service that will be used for executing scheduled asynchronous tasks. *(For future use)*	Instance of `java.util.concurrent.ScheduledExecutorService` or string representing fully qualified classname.
`fish.payara.withConfig`	The configuration for the internal JAX-RS/Jersey REST client.	Instance of `jakarta.ws.rs.core.Configuration` or a `String` representing its fully qualified classname.
`fish.payara.clientAdapter`	Implementation of client side adapter to use for intercepting JNDI lookups (see below)	Instance of `fish.payara.ejb.http.client.adapter.ClientAdapter`

The constants are also exposed as static attributes of the `fish.payara.ejb.http.client.RemoteEJBContextFactory` class.

System Properties Fallbacks

The JNDI customization options listed above as well the environment variables defined in `javax.naming.Context` (except `APPLET`) can be set by setting a system property of the same name that will act as a fallback.

That means it will not override an environment variable that is already present but would be used in the case that it is not defined when `InitialContextFactory.getInitialContext` is invoked.

Client-Side Adapter

EJB Lite connector is, in its current form, suited for invoking stateless remote EJBs. However, if you're using the connector with existing clients, those may depend on stateful interactions, like invoking stateful EJBs or accessing application server data sources and connection factories.

Client side adapters serve the purpose of emulating stateful behavior at the client and delegate the requests to stateful backend, when all information from the client is collected.

The API of client side adapters is prescribed by interface `fish.payara.ejb.http.client.adapter.ClientAdapter`:

```
public interface ClientAdapter {
    /*
     * @param jndiName jndi name requested for lookup
     * @param remoteContext naming context for remote EJB invocation
     * @return Optional.of(proxy) if adapter provides a proxy for given name,
Optional.empty() otherwise
     * @throws NamingException if downstream lookup fails, or other validation
doesn't pass
     */
    Optional<Object> makeLocalProxy(String jndiName, Context remoteContext)
        throws NamingException;
}
```

To make use of the adapter, put an instance of `ClientAdapter` into JNDI context

property `fish.payara.clientAdapter`. Every JNDI lookup will be first passed to the adapter instance in such case. If adapter returns a non-empty `Optional` value, tit will be passed back to the client.

Composing Multiple Adapters

Client side emulation might require stubbing diverse JNDI names and return types, and handling all of that in a single ClientAdapter implementation would result in not very maintainable code. Therefore, the client library offers two classes that help with composition of multiple Client adapters into the parent instance to be passed to `RemoteEJBContextFactory`:

- `CompositeClientAdapter` concatenates multiple client adapter instances, calling adapters in a defined order and returning first non-empty proxy provided by the adapters.
- `ClientAdapterCustomizer` is a decorator, that helps separate JNDI name matching from instantiation of client adapter.

Most used method of customizer is `matchPrefix`, which will only call downstream adapter when requested JNDI name matches prefix.

NOTE	This prefix is stripped before invoking the downstream adapter.

Composite Client Adapter Example

```
public class Application{
    public static void main(String[] args){
        Properties props = new Properties();
        props.put(Context.INITIAL_CONTEXT_FACTORY, RemoteEJBContextFactory
.FACTORY_CLASS);
        props.put(Context.PROVIDER_URL, "http://not.relevant/");
        props.put(RemoteEJBContextFactory.CLIENT_ADAPTER,
                CompositeClientAdapter.newBuilder()
                    .register(customize(new ConnectionFactoryAdapter()).
matchPrefix("jms/"),
                            customize(QueueAdapter.class).matchPrefix("queue/"))
                    .build()
                );
        Context context = new InitialContext(props);
    }
}
```

See API documentation of the client library for detailed description of `ClientAdapterCustomizer` methods and contracts of `CompositeClientAdapter`.

Known Limitations

Serialization Limitations

The HTTP adapter utilizes JSON-B for serialization. This places limits on types of objects that can be transmitted as method arguments or return types. The serialization can be customized by means of JSON-B annotations.

1. By default, only public properties and fields are transferred
2. Complex object graphs should form a tree and not contain cyclic references
3. Polymorphism is not supported by default

Programming Model Limitations

The HTTP adapter uses stateless HTTP requests. In its current form, it is unfit for invoking stateful objects like `@Stateful` Enterprise Java Beans, or using server resources like `DataSource` or `ConnectionFactory`.

See Also

- get-ejb-invoker-configuration
- set-ejb-invoker-configuration

Developing Java Clients

This chapter describes how to develop, assemble, and deploy remote Java clients.

NOTE	The Web Profile of Payara Server supports the Jakarta Enterprise Beans 4.0 Lite specification, which allows enterprise beans within web applications, among other features. The full Payara Server supports the entire EJB 4.0 specification. For details, see Jakarta Enterprise Beans. Accordingly, the Application Client Container is supported only in the Full Profile, not in the Web Profile. JMS resources are supported only in the Full Profile, not in the Web

> Profile. See using-jakarta-messaging.

Introducing the Application Client Container

The Application Client Container (ACC) includes a set of Java classes, libraries, and other files that are required for and distributed with Java client programs that execute in their own Java Virtual Machine (JVM). The ACC manages the execution of Jakarta EE application client components (application clients), which are used to access a variety of Jakarta EE services (such as JMS resources, EJB components, web services, security, and so on.) from a JVM **outside** Payara Server.

The ACC communicates with Payara Server using RMI-IIOP protocol and manages the details of RMI-IIOP communication using the client ORB that is bundled within it.

NOTE | Compared to other Jakarta EE containers, the ACC is lightweight.

For information about debugging application clients, see application-client-debugging.

IMPORTANT | Interoperability between application clients and Payara Servers running under different Azul Payara major versions is not supported. The same can be said if both server and client are running under different major Java versions.

ACC Security

The ACC determines when authentication is needed. This typically occurs when the client refers to an EJB component that requires authorization or when annotations in the client's `main` class trigger resource injection which, in turn, requires contact with the server's naming service.

To authenticate the end user, the ACC prompts for any required information, such as a username and password. The ACC itself provides a very simple dialog box to prompt for and read these values.

The ACC integrates with Payara Server's authentication system. It also supports SSL (Secure Socket Layer)/IIOP if configured and when necessary; see using-rmiiiop-over-

ssl.

You can provide an alternate implementation to gather authentication information, tailored to the needs of the application client. To do so, include the class to perform these duties in the application client and identify the fully-qualified name of this class in the `callback-handler` element of the `application-client.xml` descriptor for the client. The ACC uses this class instead of its default class for asking for and reading the authentication information.

The class must implement the `javax.security.auth.callback.CallbackHandler` interface. See the Jakarta EE Platform specification, section 10.2, **Application Clients: Security**, for more details.

Application clients can use a programmatic login mechanism by programmatic-login-using-the-programmaticlogin-class.

ACC Naming

The client container enables the application clients to use the Java Naming and Directory Interface (JNDI) to look up Jakarta EE services (such as JMS resources, EJB components, web services, security, and so on.) and to reference configurable parameters set at the time of deployment.

Application Client Annotation

Annotation is supported for the main class and the optional callback handler class in application clients. For more information, see "deployment-descriptors-and-annotations" in the Payara Server Application Deployment section.

Java Web Start

IMPORTANT	Java Web Start is no longer part of the Java SDK since Java 11 and is no longer a recommended solution to distribute application client binaries from the server. Using Java Web Start is not a supported solution, so consider using it at your own risk.

Java Web Start allows your application client to be easily launched and automatically

downloaded and updated. It is enabled for all application clients by default. For more information, see Using Java Web Start.

Application Client JAR File

When copying the downloaded application client to another location, make sure to include the JAR files containing the dependent classes as well. You can also use the `asadmin get-client-stubs` command to retrieve the `appclient` and all associated application JAR files and place them in another location.

Developing Clients Using the ACC

This section describes the procedure to develop, assemble, and deploy client applications using the ACC.

To Access an EJB Component from an Application Client

1. In your client code, reference the EJB component by using an `@EJB` annotation or by looking up the JNDI name as defined in the `ejb-jar.xml` file.
   For more information about naming and lookups, see accessing-the-naming-context.
   If load balancing is enabled as in Step 7 and the EJB components being accessed are in a different cluster, the endpoint list must be included in the lookup, as follows:

   ```
   corbaname:host1:port1,host2:port2,.../NameService#ejb/jndi-name
   ```

2. Define the `@EJB` annotations or the `ejb-ref` elements in the `application-client.xml` file. Define the corresponding `ejb-ref` elements in the`payara-application-client.xml` file.
   For more information on the `payara-application-client.xml` file, see "the-payara-application-client.xml-file" in the Payara Server Application Deployment section. For a general explanation of how to map JNDI names using reference elements, see mapping-references.

3. Deploy the application client and EJB component together in an application.
   To get the client JAR file, use the `--retrieve` option of the `asadmin deploy` command.
   To retrieve the stubs and ties generated during deployment, use the `asadmin get-`

`client-stubs` command.

4. Ensure that the client JAR file includes the following files:
   - A Java class to access the bean.
   - The `application-client.xml` - (optional) Jakarta EE application client deployment descriptor.
   - `payara-application-client.xml` - (optional) Payara Server specific client deployment descriptor. For information on the `payara-application-client.xml` file, see "the-payara-application-client.xml-file" in the Payara Server Application Deployment section.
   - The `MANIFEST.MF` file. This file contains a reference to the `main` class, which states the complete package prefix and class name of the Java client.
5. Prepare the client machine.
   If you are using the `appclient` script, package the Payara Server system files required to launch application clients on remote systems using the `package-appclient` script, then retrieve the application client itself using the `asadmin get-client-stubs` command.

   For more information, see Using the `package-appclient` Script .
6. To access EJB components that are residing in a remote system, make the following changes to the `sun-acc.xml` file or the `appclient` script.
   - Define the `target-server` element's `address` and `port` attributes to reference the remote server machine and its ORB port. See "target-server" in the Payara Server Application Deployment section.
   - Use the `-targetserver` option of the `appclient` script to reference the remote server machine and its ORB port.For more information, see Running an Application Client Using the `appclient` Script.

     To determine the ORB port on the remote server, use the `asadmin get` command. For example:

```
asadmin --host <remote-host> get server-config.iiop-service.iiop-
listener.iiop-listener1.port
```

7. To set up load balancing and failover of remote EJB references, define at least two `target-server` elements in the `sun-acc.xml` file or the `appclient` script. If the Payara Server instance on which the application client is deployed participates in a cluster or deployment group, the ACC finds all currently active IIOP endpoints in the cluster automatically. However, a client should have at least two endpoints specified for bootstrapping purposes, in case one of the endpoints has failed.

   The `target-server` elements in the `sun-acc.xml` file specify one or more IIOP endpoints used for load balancing. The `address` attribute is an IPv4 address or host name, and the `port` attribute specifies the port number.

   See "client-container" in the Payara Server Application Deployment section.

   The `--targetserver` option of the `appclient` script specifies one or more IIOP endpoints used for load balancing. For more information, see Running an Application Client Using the `appclient` Script.

Next Steps

- For instructions on running the application client, see Using Java Web Start or Running an Application Client Using the `appclient` Script.
- For more information about RMI-IIOP load balancing and failover, see "rmi-iiop-load-balancing-and-failover" in the Payara Server High Availability section.

To Access a JMS Resource from an Application Client

1. Create a JMS client.

   For detailed instructions on developing a JMS client, see "Jakarta Messaging Examples" in The Jakarta EE Tutorial.

2. Next, configure a JMS resource in Payara Server.

For information on configuring JMS resources, see "administering-jms-connection-factories-and-destinations" in the Payara Server General Administration section.

3. Define the `@Resource` or `@Resources` annotations or the `resource-ref` elements in the `application-client.xml` file. Define the corresponding `resource-ref` elements in the `payara-application-client.xml` file.
   For more information on the `payara-application-client.xml` file, see "the-payara-application-client.xml-file" in the Payara Server Application Deployment section. For a general explanation of how to map JNDI names using reference elements, see mapping-references.
4. Ensure that the client JAR file includes the following files:
   - A Java class to access the resource.
   - `application-client.xml` - (optional) Jakarta EE application client deployment descriptor.
   - `payara-application-client.xml` - (optional) Payara Server specific client deployment descriptor. For information on the `payara-application-client.xml` file, see "the-payara-application-client.xml-file" in the Payara Server Application Deployment section.
   - The `MANIFEST.MF` file. This file contains a reference to the `main` class, which states the complete package prefix and class name of the Java client.
5. Prepare the client machine.

   TIP | This step is not needed if the client and server machines are the same.

   If you are using the `appclient` script, package the Payara Server system files required to launch application clients on remote systems using the `package-appclient` script, then retrieve the application client itself using the `asadmin get-client-stubs` command.

   For more information, see Using the `package-appclient` Script

6. Run the application client.

   See Using Java Web Start or Running an Application Client Using the `appclient` Script.

Using Java Web Start

IMPORTANT	Java Web Start is no longer part of the JDK since Java 11 and is no longer a recommended solution to distribute application client binaries. To use Java Web Start, you'll have to manually install an implementation on the local machine where the Application client is intended to run. *Using Java Web Start is not a supported solution, so consider using it at your own risk.*

Java Web Start allows your application client to be easily launched and automatically downloaded and updated.

Enabling and Disabling Java Web Start

NOTE	Java Web Start is enabled for all application clients by default.

The application developer or deployer can specify that Java Web Start is always disabled for an application client by setting the value of the `eligible` element to `false` in the `payara-application-client.xml` file.

The server administrator can disable Java Web Start for a previously deployed eligible application client using the `asadmin set` command.

To disable Java Web Start for all eligible application clients in an application, use the following command:

```
asadmin set applications.application.app-name.property.java-web-start-enabled
="false"
```

To disable Java Web Start for a stand-alone eligible application client, use the following command:

```
asadmin set applications.application.module-name.property.java-web-start-enabled
="false"
```

Setting `java-web-start-enabled="true"` re-enables Java Web Start for an eligible application client.

Downloading and Launching an Application Client

If Java Web Start is enabled for your deployed application client, you can launch it for testing. Simply click on the *Launch* button next to the application client or application's listing on the App Client Modules page in the Administration Console.

On other machines, you can download and launch the application client using Java Web Start in the following ways:

- Using a web browser, directly enter the URL for the application client. See The Application Client URL.
- Click on a link to the application client from a web page.
- Use the Java Web Start command `javaws`, specifying the URL of the application client as a command line argument.
- If the application has previously been downloaded using Java Web Start, you have additional alternatives:
  - Use the desktop icon that Java Web Start created for the application client. When Java Web Start downloads an application client for the first time it asks you if such an icon should be created.
  - Use the Java Web Start control panel to launch the application client.

Whenever you launch an application client, Java Web Start contacts the server to see if a newer client version is available. This means you can redeploy an application client without having to worry about whether client machines have the latest version.

The Application Client URL

The default URL for an application or module generally is as follows:

```
http(s)://host:port/context-root
```

The default URL for a stand-alone application client module is as follows:

```
http://host:port/appclient-module-id
```

The default URL for an application client module embedded within an application is as follows.

NOTE | The relative path to the application client JAR file is included.

```
http://host:port/application-id/appclient-path
```

If the context-root, `appclient-module-id`, or `application-id` is not specified during deployment, the name of the JAR or EAR file without the extension is used.

If the application client module or application is not in JAR or EAR file format, an `appclient-module-id` or `application-id` is generated.

Regardless of how the context-root or id is determined, it is written to the server log when you deploy the application. For details about naming, see "naming-standards" in the Payara Server Application Deployment section.

To set a different URL for an application client, use the `context-root` sub-element of the `java-web-start-access` element in the `payara-application-client.xml` file. This overrides the `appclient-module-id` or `application-id`.

You can also pass arguments to the ACC or to the application client's `main` method as query parameters in the URL. If multiple application client arguments are specified, they are passed in the order specified.

A question mark separates the context root from the arguments. Ampersands (`&`)

separate the arguments and their values. Each argument and each value must begin with the `arg=` prefix.

Here is an example URL with a `-color` argument for a stand-alone application client. The `-color` argument is passed to the application client's `main` method:

```
http://localhost:8080/testClient?arg=-color&arg=red
```

> **TIP** If you are using the `javaws` URL command to launch Java Web Start with a URL that contains arguments, enclose the URL in double quotes (`"`) to avoid breaking the URL at the ampersand (`&`) symbol.

Ideally, you should build your production application clients with user-friendly interfaces that collect information which might otherwise be gathered as command-line arguments. This minimizes the degree to which users must customize the URLs that launch application clients using Java Web Start.

Command-line argument support is useful in a development environment and for existing application clients that depend on it.

Signing JAR Files Used in Java Web Start

Java Web Start enforces a security sandbox. By default, it grants any application, including application clients, only minimal privileges. Because Java Web Start applications can be so easily downloaded, Java Web Start provides protection from potentially harmful programs that might be accessible over the network.

If an application requires a higher privilege level than the sandbox permits, the code that needs privileges must be in a JAR file that was signed.

When Java Web Start downloads such a signed JAR file, it displays information about the certificate that was used to sign the JAR if that certificate is not trusted. It then asks you whether you want to trust that signed code. If you agree, the code receives elevated permissions and runs. If you reject the signed code, Java Web Start does not start the downloaded application.

Your first Java Web Start launch of an application client is likely to involve this prompting because by default Payara Server uses a self-signed certificate that is not linked to a trusted authority.

NOTE	Payara Server serves two types of signed JAR files in response to Java Web Start requests. One type is a JAR file installed as part of the Payara Server, which starts an application client during a Java Web start launch: as-install`/lib/gf-client.jar`.

The other type is a generated application client JAR file. As part of deployment, the Payara Server generates a new application client JAR file that contains classes, resources, and descriptors needed to run the application client on end-user systems. When you deploy an application with the `asadmin deploy` command's `--retrieve` option, use the `asadmin get-client-stubs` command, or select the **Generate RMIStubs** option from the EJB Modules deployment page in the Administration Console, this is one of the JAR files retrieved to your system. Because application clients need access beyond the minimal sandbox permissions to work in the Java Web Start environment, the generated application client JAR file must be signed before it can be downloaded to and executed on an end-user system.

A JAR file can be signed automatically or manually.

Automatically Signing JAR Files

Payara Server automatically creates a signed version of the required JAR file if none exists. When a Java Web Start request for the `gf-client.jar` file arrives, the Payara Server looks for the `domain-dir/java-web-start/gf-client.jar` file. When a request for an application's generated application client JAR file arrives, the Payara Server looks in the `domain-dir/java-web-start/app-name` directory for a file with the same name as the generated JAR file created during deployment.

In either case, if the requested signed JAR file is absent or older than its unsigned counterpart, Payara Server creates a signed version of the JAR file automatically and deposits it in the relevant directory. Whether Payara Server just signed the JAR file or

not, it serves the file from the `domain-dir/java-web-start` directory tree in response to the Java Web Start request.

To sign these JAR files, by default Payara Server uses its self-signed certificate. When you create a new domain by using the `asadmin create-domain` command, the Payara Server creates a self-signed certificate and adds it to the domain's key store (the same applies to the default domains included in the server's installation bundle).

WARNING	A self-signed certificate is generally untrustworthy because no certification authority vouches for its authenticity.

The automatic signing feature uses the same certificate to create all required signed JAR files.

To minimize impacts to application users, all Java Web Start applications should be signed with a trusted certificate instead of the Payara Server self-signed certificate.

If you use the Payara Server Java Web Start feature or deploy applications that provide their own Java Web Start applications, perform the following steps:

1. Obtain a trusted certificate from a certification authority if your organization does not already have one.
2. Stop the Payara Server domain and corresponding instances, clusters or deployment groups.
3. Replace the self-signed certificate with the trusted certificate by importing the trusted certificate into the keystore using the `s1as` alias.

   By default, the keystore is located at `domain-dir/config/keystore.p12`.

   For more information about importing a trusted certificate into the domain keystore, see "administering-jsse-certificates" in the Payara Server Security Guide.
4. Delete any signed JARs already generated by Payara Server:

a. At the command prompt, type:

```
rm -rf ${domain-dir}/java_web_start
```

b. For each application that contains an application client launched using Java Web Start, type:

```
rm -rf ${domain-dir}/generated/xml/app-name/signed
```

5. Restart the Payara Server domain and corresponding targets.

After you perform these steps, the first time a user launches an application client on their system, Java Web Start detects that the server's signed JARs are newer than those cached on the user's system and downloads them again. This happens on the first launch only, regardless of the client.

Even though the application client is now signed using a trusted certificate, users will again be asked whether to trust the downloaded application and can choose to skip that prompt for future launches.

Using the `jar-signing-alias` Deployment Property

The `asadmin deploy` command property `jar-signing-alias` specifies the alias for the security certificate with which the application client container JAR file is signed.

Java Web Start won't execute code requiring elevated permissions unless it resides in a JAR file signed with a certificate that the user's system trusts. For your convenience, Payara Server signs the JAR file automatically using the self-signed certificate from the domain, `s1as`. Java Web Start then asks the user whether to trust the code and displays the server's certificate information.

To sign this JAR file with a different certificate, first add the certificate to the domain keystore. You can use a certificate from a trusted authority, which avoids the Java Web Start prompt. To add a certificate to the domain keystore, see "administering-jsse-certificates" in the Payara Server Security Guide.

Next, deploy your application using the `jar-signing-alias` property. For example:

```
asadmin deploy --property jar-signing-alias=MyAlias MyApp.ear
```

Error Handling

When an application client is launched using Java Web Start, any error that the application client logic does not catch and handle is written to `System.err` and displayed in a dialog box.

This display appears if an error occurs even before the application client logic receives control. It also appears if the application client code does not catch and handle errors itself.

Vendor Icon, Splash Screen, and Text

To specify a vendor-specific icon, splash screen, text string, or a combination of these for Java Web Start download and launch screens, use the `vendor`element in the `payara-application-client.xml` file. The complete format of this element's data is as follows:

```
<vendor>icon-image-URI::splash-screen-image-URI::vendor-text</vendor>
```

The following example vendor element contains an icon, a splash screen, and a text string:

```
<vendor>images/icon.jpg::otherDir/splash.jpg::MyCorp, Inc.</vendor>
```

The following example vendor element contains an icon and a text string:

```
<vendor>images/icon.jpg::MyCorp, Inc.</vendor>
```

The following example vendor element contains a splash screen and a text string; note the initial double colon:

```
<vendor>::otherDir/splash.jpg::MyCorp, Inc.</vendor>
```

The following example vendor element contains only a text string:

```
<vendor>MyCorp, Inc.</vendor>
```

NOTE | The default value is the text string `Application Client`.

You can also specify a vendor-specific icon, splash screen, text string, or a combination by using a custom JNLP file; see Creating a Custom JNLP File.

Creating a Custom JNLP File

You can partially customize the Java Network Launching Protocol (JNLP) file that Payara Server uses for Java Web Start.

For more information about JNLP, see the JNLP Specification and API Documentation.

Specifying the JNLP File in the Deployment Descriptor

To specify a custom JNLP file for Java Web Start, use the `jnlp-doc` element in the `payara-application-client.xml` file. If `none` is specified, a default JNLP file is generated.

The value of the `jnlp-doc` element is a relative path with the following format:

```
[path-to-JAR-in-EAR!]path-to-JNLP-in-JAR
```

The default `path-to-JAR-in-EAR` is the current application client JAR file. For example, if the JNLP file is in the application client JAR file at `custom/myInfo.jnlp`, the element value would look like this:

```
<java-web-start-access>
   <jnlp-doc>custom/myInfo.jnlp</jnlp-doc>
</java-web-start-access>
```

If the application client is inside an EAR file, you can place the custom JNLP file inside another JAR file in the EAR.

For example, if the JNLP file is in a JAR file at `other/myLib.jar`, the element value

would look like this, with an exclamation point (`!`) separating the path to the JAR from the path in the JAR:

```
<java-web-start-access>
   <jnlp-doc>other/myLib.jar!custom/myInfo.jnlp</jnlp-doc>
</java-web-start-access>
```

For more information about the `payara-application-client.xml` file, see the Payara Server Application Deployment section.

Referring to JAR Files from the JNLP File

As with any JNLP document, the custom JNLP file can refer to JAR files the application client requires.

NOTE

Do not specify every JAR on which the client depends. Payara Server automatically handles JAR files that the Jakarta EE specification requires to be available to the application client. This includes JAR files listed in the application client JAR file's manifest `Class-Path` and JAR files in the EAR file's library directory (if any) and their transitive closures. The custom JNLP file should specify only those JAR files the client needs that Payara Server would not otherwise include.

Package these JAR files in the EAR file, as with any JAR file required by an application client. Use relative URIs in the `<jar href="…">` and `<nativelib href="…">` elements to point to these JAR files. The codebase that Payara Server assigns for the final client JNLP file corresponds to the top level of the EAR file.

Therefore, relative `href` references correspond directly to the relative path to the JAR files within the EAR file.

Neither the Jakarta EE specification nor Payara Server supports packaging JAR files inside the application client JAR file itself. Nothing prevents this, but Payara Server does no special processing of such JAR files. They do not appear in the runtime class path, and they cannot be referenced from the custom JNLP file as well.

Referring to Other JNLP Files

The JNLP file can also refer to other custom JNLP files using `<extension href="…"/>` elements. To be consistent with relative `href` references to JAR files, the relative `href` references to JNLP files are resolved within the EAR file. You can place these JNLP files directly in the EAR file or inside JAR files that the EAR file contains.

Use one of these formats for these `href` references:

```
[path-to-JAR-in-EAR!]path-to-JNLP-in-JAR

path-to-JNLP-in-EAR
```

Note that these formats are not equivalent to the format of the `jnlp-doc` element in the `payara-application-client.xml` file.

WARNING	These formats follow the standard entry-within-a-JAR URI syntax and semantics. Support for this syntax comes from the automated Java Web Start support in Payara Server. This is not a feature of Java Web Start or the JNLP standard.

Combining Custom and Automatically Generated Content

Payara Server recognizes these types of content in the JNLP file:

Owned: Payara Server owns the content and ignores any custom content **Merged**: Automatically generated content and custom content are merged **Defaulted**: Custom content is used if present, otherwise default content is provided

You can compose a complete JNLP file and package it with the application client. Payara Server then combines it with its automatically generated JNLP file.

You can also provide content that only adds to or replaces what Payara Server generates. The custom content must conform to the general structure of the JNLP format so that Payara Server can properly place it in the final JNLP file.

For example, to specify a single native library to be included only for Windows systems, the new element to add might be as follows:

```
<nativelib href="windows/myLib.jar"/>
```

However, you must indicate where in the overall document this element belongs. The actual custom JNLP file should look like this:

```
<jnlp>
   <resources os="Windows">
      <nativelib href="windows/myLib.jar"/>
   </resources>
</jnlp>
```

Payara Server provides default `<information>` and `<resources>` elements, without specifying attributes such as `os`, `arch`, `platform`, or `locale`. Payara Server merges its own content within those elements with custom content under those elements. Further, you can provide your own `<information>` and `<resources>` elements (and fragments within them) that specify at least one of these attributes.

In general, you can perform the following customizations:

- Override the Payara Server defaults for the child elements of `<information>` elements that have no attribute settings for `os`, `arch`, `platform`, and `locale`.

  Among these child elements are `<title>`, `<vendor>`, `<description>`, `<icon>`, and so on.

- Add `<information>` elements with `os`, `arch`, `platform`, or `locale` settings. You can also add child elements.
- Add child elements of `<resources>` elements that have no attribute settings for `os`, `arch`, or `locale`.

  Among these child elements are `<jar>`, `<property>`, `<nativelib>`, and so on.

  You can also customize attributes of the `<java>` child element.

- Add `<resources>` elements that specify at least one of `os`, `arch`, or `locale`. You can also add child elements.

This flexibility allows you to add JAR files to the application (including platform-specific native libraries) and set properties to control the behavior of your application clients.

The following tables provide more detail about what parts of the JNLP file you can add to and modify.

Table 193. Owned JNLP File Content

JNLP File Fragment	Description
`<jnlp codebase="xxx">...</jnlp>`	Payara Server controls this content for application clients packaged in EAR files. The developer controls this content for application clients packaged in WAR files.
`<jnlp href="xxx">...</jnlp>`	Payara Server controls this content for application clients packaged in EAR files. The developer controls this content for application clients packaged in WAR files.

`<jnlp>`
`  <security>...</security>`
`</jnlp>`
	Payara Server must control the permissions requested for each JNLP file. All permissions are needed for the main file, which launches the ACC. The permissions requested for other JNLP documents depend on whether the JAR files referenced in those documents are signed.

`<jnlp>`
`  <application-desc>`
`  <argument> ... </argument>`
`  </application-desc>`
`</jnlp>`
	Payara Server sets the `main-class` and the arguments to be passed to the client.

Table 194. Defaulted JNLP File Content

JNLP File Fragment	Description
`<jnlp spec="xxx">...</jnlp>`	Specifies the JNLP specification version.

`<jnlp>`
`  <information>...</information>`
`</jnlp>`
	Specifies the application title, vendor, home page, various description text values, icon images, and whether offline execution is allowed.

JNLP File Fragment	Description

`<jnlp>`
`  <resources>`
`  <java version="xxx" java-vm-args="yyy"></java>`
`  </resources>`
`</jnlp>`
	Specifies the Java SE version or selected VM parameter settings.

Table 195. Merged JNLP File Content

JNLP File Fragment	Description

`<jnlp>`
`  <information os="" platform="" arch="" locale=""></information>`
`</jnlp>`
	You can specify one or more of the `os`, `arch`, `platform`, and `locale` attributes for the `<information>` element. You can also specify child elements; Payara Server provides no default children.

`<jnlp>`
`  <resources os="" platform="" arch="" locale=""></resources>`
`</jnlp>`
	You can specify one or more of the `os`, `arch`, `platform`, and `locale` attributes for the `<resources>` element. You can also specify child elements; Payara Server provides no default children.

`<jnlp>`
`  <resources>`
`  <jar>...</jar>`
`  </resources>`
`</jnlp>`
	Adds JAR files to be included in the application to the JAR files provided by Payara Server.

`<jnlp>`
`  <resources>`
`  <nativelib>...</nativelib>`
`  </resources>`
`</jnlp>`
	Adds native libraries to be included in the application. Each entry in a JAR listed in a `<nativelib>` element must be a native library for the correct platform. The full syntax of the `<nativelib>` element lets the developer specify the platform for that native library.

`<jnlp>`
`  <resources>`
`  <property></property>`
`  </resources>`
`</jnlp>`
	Adds system properties to be included in the application to the system properties defined by Payara Server.

`<jnlp>`
`  <resources>`
`  <extension>...</extension>`
`  </resources>`
`</jnlp>`
	Specifies another custom JNLP file.

JNLP File Fragment	Description

`<jnlp>`
`  <component-desc></component-desc>`
`</jnlp>`
	Includes another custom JNLP file that specifies a component extension.

`<jnlp>`
`  <installer-desc></installer-desc>`
`</jnlp>`
	Includes another custom JNLP file that specifies an installer extension.

Using the Embeddable ACC

You can embed the ACC into your application client. If you place the `as-install/lib/gf-client.jar` file in your runtime classpath, your application creates the ACC after your application code has started, then requests that the ACC start the application client portion. The basic model for coding with the embeddable ACC is as follows:

1. Create a builder object.
2. Operate on the builder to configure the ACC.
3. Obtain a new ACC instance from the builder.
4. Present a client archive or class to the ACC instance.
5. Start the client running within the newly created ACC instance.

Your code should follow this general pattern:

```
// one TargetServer for each ORB endpoint for bootstrapping
TargetServer[] servers = ...;

// Get a builder to set up the ACC
AppClientContainer.Builder builder = AppClientContainer.newBuilder(servers);

// Fine-tune the ACC's configuration. Note ability to "chain" invocations.
builder.callbackHandler("com.acme.MyHandler").authRealm("myRealm"); // Modify
config

// Get a container for a client.
URI clientURI = ...; // URI to the client JAR
AppClientContainer acc = builder.newContainer(clientURI);

or

Class mainClass = ...;
AppClientContainer acc = builder.newContainer(mainClass);
```

```
// In either case, start the client running.
String[] appArgs = ...;
acc.startClient(appArgs); // Start the client

...

acc.close(); // close the ACC
```

The ACC loads the application client's `main` class, performs any required injection, and transfers control to the `static main` method. The ACC's `run` method returns to the calling application as soon as the client's `main` method returns to the ACC.

If the application client's `main` method starts any asynchronous activity, that work continues after the ACC returns. The ACC has no knowledge of whether the client's `main` method triggers asynchronous work.

Therefore, if the client causes work on threads other than the calling thread, and if the embedding application needs to know when the client's asynchronous work completes, the embedding application and the client must agree on how this happens.

The ACC's shutdown handling is invoked from the ACC's `close` method. The calling application can invoke `acc.close()` to close down any services started by the ACC.

If the application client code started any asynchronous activity that might still depend on ACC services, invoking `close` before that asynchronous activity completes could cause unpredictable and undesirable results.

The shutdown handling is also run automatically at VM shutdown if the code has not invoked `close` before then.

WARNING	The ACC does not prevent the calling application from creating or running more than one ACC instance during a single execution of the application either serially or concurrently. However, other services used by the ACC (transaction manager, security, ORB, and so on) might or might not support such serial or concurrent reuse.

Running an Application Client Using the `appclient` Script

To run an application client, you can launch the ACC using the `appclient` script, whether Java Web Start is enabled. This script is located in the `as-install/glassfish/bin` directory.

Using the `package-appclient` Script

You can package the Payara Server system files required to launch application clients on remote systems into a single JAR file using the `package-appclient` script. This script is located in the `as-install/glassfish/bin` directory.

The `client.policy` File

The `client.policy` file is the policy file used by the application client. Each application client has a `client.policy` file.

The default policy file limits the permissions of Jakarta EE deployed application clients to the minimal set of permissions required for these applications to operate correctly. If an application client requires more than this default set of permissions, edit the `client.policy` file to add the custom permissions that your application client needs.

For more information about the permissions you can set in the `client.policy` file, see Permissions in the JDK.

Using RMI/IIOP over SSL

You can configure RMI/IIOP over SSL in two ways: using a username and password, or using a client certificate.

To use a username and password, configure the `ior-security-config` element in the `payara-ejb-jar.xml` file. The following configuration establishes SSL between an application client and an EJB component using a username and password.

The user has to log in to the ACC using either the `sun-acc.xml` mechanism or the programmatic-login-using-the-programmaticlogin-class mechanism.

```
<ior-security-config>
  <transport-config>
    <integrity>required</integrity>
    <confidentiality>required</confidentiality>
```

```
    <establish-trust-in-target>supported</establish-trust-in-target>
    <establish-trust-in-client>none</establish-trust-in-client>
  </transport-config>
  <as-context>
    <auth-method>username_password</auth-method>
    <realm>default</realm>
    <required>true</required>
  </as-context>
 <sas-context>
    <caller-propagation>none</caller-propagation>
 </sas-context>
</ior-security-config>
```

To use a client certificate, configure the `ior-security-config` element in the `payara-ejb-jar.xml` file.

The following configuration establishes secure transport between an application client and an EJB component using a client certificate.

```
<ior-security-config>
  <transport-config>
    <integrity>required</integrity>
    <confidentiality>required</confidentiality>
    <establish-trust-in-target>supported</establish-trust-in-target>
    <establish-trust-in-client>required</establish-trust-in-client>
  </transport-config>
  <as-context>
    <auth-method>none</auth-method>
    <realm>default</realm>
    <required>false</required>
  </as-context>
  <sas-context>
    <caller-propagation>none</caller-propagation>
  </sas-context>
</ior-security-config>
```

To use a client certificate, you must also specify the system properties for the keystore and truststore to be used in establishing SSL.

To use SSL with the Application Client Container (ACC), you need to set these system properties in one of the following ways:

- Use the new syntax of the `appclient` script and specify the system properties as JVM options. See Running an Application Client Using the `appclient` Script.
- Set the environment variable `VMARGS` in the shell. For example, in the `ksh` or `bash` shell, the command to set this environment variable would be as follows:

```
export VMARGS="-Djavax.net.ssl.keyStore=${keystore.db.file}
-Djavax.net.ssl.trustStore=${truststore.db.file}
-Djavax.net.ssl.keyStorePassword=${ssl.password}
-Djavax.net.ssl.trustStorePassword=${ssl.password}"
```

Connecting to a Remote EJB Module through a Firewall

To deploy and run an application client that connects to an EJB module on a Payara Server instance that is behind a firewall, you must set ORB Virtual Address Agent Implementation (**ORBVAA**) options. Use the `asadmin create-jvm-options` command as follows:

```
asadmin create-jvm-options -Dcom.sun.corba.ee.ORBVAAHost=public-IP-adress
asadmin create-jvm-options -Dcom.sun.corba.ee.ORBVAAPort=public-port
asadmin create-jvm-options
-
Dcom.sun.corba.ee.ORBUserConfigurators.com.sun.corba.ee.impl.plugin.hwlb.VirtualAdd
ressAgentImpl=x
```

Set the `ORBVAAHost` and `ORBVAAPort` options to the host and port of the public address. The `ORBUserConfigurators` option tells the ORB to create an instance of the `VirtualAddressAgentImpl` class and invoke the `configure` method on the resulting object, which must implement the com.sun.corba.ee.spi.orb.ORBConfigurator interface.

NOTE

The `ORBUserConfigurators` value doesn't matter.Together, these options create an ORB that in turn creates `Object` references (the underlying implementation of remote EJB references) containing the public address, while the ORB listens on the private address specified for the IIOP port in the Payara Server configuration.

Specifying a Splash Screen

Java SE offers splash screen support, either through a Java command-line option or a manifest entry in the application's JAR file.

To take advantage of this Java SE feature in your application client, you can do one of the following:

- Create the appclient JAR file so that its manifest contains a `SplashScreen-Image` entry that specifies the path to the image in the client.The `java` command displays the splash screen before starting the ACC or your client, just as with any Java application.
- Use the new `appclient … -jar` launch format, using the `-splash` command-line option at runtime or the `SplashScreen-Image` manifest entry at development time.

  See Running an Application Client Using the `appclient` Script.
- In the environment that runs the `appclient` script, set the `VMOPTS` environment variable to include the `-splash` option before invoking the `appclient` script to launch the client.
- Build an application client that uses the embeddable ACC feature and specify the splash screen image using one of the following:
  - The `-splash` option of the `java` command
  - `SplashScreen-Image` in the manifest for your program (not the manifest for the application client)

    See Using the Embeddable ACC.

During application (EAR file) deployment, the Payara Server generates facade JAR files, one for the application and one for each application client in the application. During application client module deployment Payara Server generates a single facade JAR for the application client.

The `appclient` script supports splash screens inside the application client JAR only if you launch an application client facade or appclient client JAR. If you launch the facade for an application or the un-deployed application itself, the `appclient` script cannot take advantage of the splash screen feature.

Setting Login Retries

You can set a JVM option using the `appclient` script that determines the number of login retries allowed. This option is

`-Dorg.glassfish.appclient.acc.maxLoginRetries=n` where `n` is a positive integer. The default number of retries is `3`.

This retry loop happens when the ACC attempts to perform injection if you annotated the client's `main` class (for example, using `@Resource`).

If instead of annotations your client uses the `InitialContext` explicitly to look up remote resources, the retry loop does not apply.

TIP	In this case, you could write logic to catch an exception around the lookup and retry explicitly.

Using Libraries with Application Clients

The Libraries field in the Administration Console's deployment page and the `--libraries` option of the `asadmin deploy` command do not apply to application clients.

Neither do the `as-install/lib`, `domain-dir/lib`, and `domain-dir/lib/classes` directories comprising the Common Class Loader.

These apply only to applications and modules deployed to the server. For more information, see class-loaders.

To use libraries with an application client, package the application client in an application (EAR file). Then, either place the libraries in the `/lib` directory of the EAR file or specify their location in the application client JAR file's manifest `Class-Path`.

Developing Clients without the ACC

This section describes the procedure to create, assemble, and deploy a Java-based client that is not packaged using the Application Client Container (ACC).

To Access an EJB Component from a Stand-Alone Client

1. In your client code, instantiate the `InitialContext`:

```
InitialContext ctx = new InitialContext();
```

It is not necessary to explicitly instantiate a naming context that points to the `CosNaming` service.

2. In the client code, look up the home object by specifying the JNDI name of the home object. Here is an EJB 2.x example:

```
Object ref = ctx.lookup("jndi-name");
BeanAHome = (BeanAHome)PortableRemoteObject.narrow(ref,BeanAHome.class);
```

Here is an EJB 3.x+ example:

```
BeanRemoteBusiness bean =(BeanRemoteBusiness) ctx.lookup
("com.acme.BeanRemoteBusiness");
```

If load balancing is enabled as in Step 6 and the EJB components being accessed are in a different cluster or deployment group, the endpoint list must be included in the lookup, as follows:

```
corbaname:host1:port1,host2:port2,.../NameService#ejb/jndi-name
```

3. Deploy the EJB component to be accessed.

For more information on deployment, see "about-deployment-tools" in the Payara Server Application Deployment section.

4. Copy the `as-install/lib/gf-client.jar` file to the client machine and include it in the classpath on the client side.

The `gf-client.jar` file references the Payara Server JAR files in its `MANIFEST.MF` file. If there is no Payara Server installation on the client machine, you must also copy the `as-install/modules` directory to the client machine and maintain its directory structure relative to the `as-install/lib/gf-client.jar` file. Or you can use the `package-appclient` script; see Using the `package-appclient` Script.

5. To access EJB components that are residing in a remote system, set the following system properties for the Java Virtual Machine startup options:

```
-Dorg.omg.CORBA.ORBInitialHost=${ORBhost}
-Dorg.omg.CORBA.ORBInitialPort=${ORBport}
```

Here `ORBhost` is the Payara Server hostname and `ORBport` is the ORB port number (default is `3700` for the default server instance, named `server`).

You can use the `asadmin get` command to get the IIOP port numbers. For example:

```
asadmin get "configs.config.server-config.iiop-service.iiop-listener.orb-listener-1.*"
```

6. To set up load balancing and remote EJB reference failover, define the `endpoints` property as follows:

   ```
   -Dcom.sun.appserv.iiop.endpoints=host1:port1,host2:port2,...
   ```

   The `endpoints` property specifies a comma-separated list of one or more IIOP endpoints used for load balancing. An IIOP endpoint is in the form `host:port`, where the host is an IPv4 address or host name, and the port specifies the port number.

   If the `endpoints` list is changed dynamically in the code, the new list is used only if a new `InitialContext` is created.

7. Make sure the `etc/hosts` file on the client machine maps the Payara Server hostname and external IP address correctly.
8. Run the stand-alone client.

   As long as the client environment is set appropriately and the JVM is compatible, you merely need to run the application's `main` class.

To Access an EJB Component from a Server-Side Module

A server-side module can be a servlet, another EJB component, or another type of module.

1. In your module code, instantiate the `InitialContext`:

```
InitialContext ctx = new InitialContext();
```

It is not necessary to explicitly instantiate a naming context that points to the `CosNaming` service. To set up load balancing and remote EJB reference failover, define the `endpoints` property as follows:

```
Hashtable env = new Hashtable();
env.put("com.sun.appserv.iiop.endpoints","host1:port1,host2:port2,...");
InitialContext ctx = new InitialConext(env);
```

The `endpoints` property specifies a comma-separated list of one or more IIOP endpoints used for load balancing. An IIOP endpoint is in the form `host:port`, where the host is an IPv4 address or host name, and the port specifies the port number.

You can use the `asadmin get` command to get the IIOP port numbers. If the `endpoints` list is changed dynamically in the code, the new list is used only if a new `InitialContext` is created. For example:

```
asadmin get "configs.config.server-config.iiop-service.iiop-listener.orb-
listener-1.*"
```

2. In the module code, look up the home object by specifying the JNDI name of the home object. Here is an EJB 2.x example:

```
Object ref = ctx.lookup("jndi-name");
BeanAHome = (BeanAHome)PortableRemoteObject.narrow(ref,BeanAHome.class);
```

Here is an EJB 3.x+ example:

```
BeanRemoteBusiness bean =(BeanRemoteBusiness) ctx.lookup
("com.acme.BeanRemoteBusiness");
```

If load balancing is enabled as in Step 1 and the EJB components being accessed are in a different cluster or deployment group, the endpoint list must be included in

the lookup, as follows:

```
corbaname:host1:port1,host2:port2,.../NameService#ejb/jndi-name
```

For more information about naming and lookups, see accessing-the-naming-context.

3. Deploy the EJB component to be accessed. For more information on deployment, see "about-deployment-tools" in the Application Deployment section.
4. To access EJB components that are residing in a remote system, set the following system properties for the Java Virtual Machine startup options:

```
-Dorg.omg.CORBA.ORBInitialHost=${ORBhost}
-Dorg.omg.CORBA.ORBInitialPort=${ORBport}
```

Here `ORBhost` is the Application Server hostname and `ORBport` is the ORB port number (default is `3700` for the default server instance, named `server`).

5. Deploy the module.

For more information on deployment, see "about-deployment-tools" in Payara Server Application Deployment section.

To Access a JMS Resource from a Stand-Alone Client

1. Create a JMS client.

For detailed instructions on developing a JMS client, see "Jakarta Messaging Examples" in The Jakarta EE Tutorial.

2. Configure a JMS resource on Payara Server.

For information on configuring JMS resources, see "administering-jms-connection-factories-and-destinations" in the Payara Server General Administration section.

3. Copy the following JAR files to the client machine and include them in the classpath on the client side:

- `gf-client.jar` - available at `as-install/lib`
- `imqjmsra.jar` - available at `as-install/lib/install/applications/jmsra`

The `gf-client.jar` file references the Payara Server JAR files in its `MANIFEST.MF` file. If there is no Payara Server installation on the client machine, you must also copy the `as-install/modules` directory to the client machine and maintain its directory structure relative to the as-install`/lib/gf-client.jar` file. Or you can use the `package-appclient` script; see Using the `package-appclient` Script.

4. To access EJB components that are residing in a remote system, set the following system properties for the Java Virtual Machine startup options:

```
-Dorg.omg.CORBA.ORBInitialHost=${ORBhost}
-Dorg.omg.CORBA.ORBInitialPort=${ORBport}
```

Here `ORBhost` is the Application Server hostname and `ORBport` is the ORB port number (default is `3700` for the default server instance, named `server`). You can use the `asadmin get` command to get the IIOP port numbers. For example:

```
asadmin get "configs.config.server-config.iiop-service.iiop-listener.orb-listener-1.*"
```

5. Run the stand-alone client.

As long as the client environment is set appropriately and the JVM is compatible, you merely need to run the application's `main` class.

Developing Connectors

NOTE | This section strictly applies to Payara Server Full Profile.

This chapter describes Payara Server support for the Jakarta Connectors specification.

Jakarta Connectors provides a Java solution to the problem of connectivity between

multiple application servers and existing enterprise information systems (EIS). By using Jakarta Connectors, EIS vendors no longer need to customize their product for each application server. Application server vendors who conform to Jakarta Connectors do not need to write custom code to add connectivity to a new EIS.

This chapter uses the terms **connector** and **resource adapter** interchangeably. Both terms refer to a resource adapter module that is developed in conformance with the Jakarta Connectors Specification.

IMPORTANT	If you installed the Web Profile, connector modules that use only outbound communication features and work-management that does not involve inbound communication features are supported. Other connector features are supported only in Full Profile.

For more information about connectors, see "Resource Adapters and Contracts" in the Jakarta EE Tutorial.

Connector Support in Payara Server

Payara Server supports the development and deployment of resource adapters that are compatible with the Jakarta Connector specification in the following versions:

- The Connector 1.0 specification defines the outbound connectivity system contracts between the resource adapter and Payara Server.
- The Connector 1.5 specification introduces major additions in defining system level contracts between the Payara Server and the resource adapter with respect to inbound connectivity, life cycle management, and thread management.
- The Connector 1.6 specification introduces further additions in defining system level contracts between the Payara Server and the resource adapter with respect to the following: **Generic work context contract**: A generic contract that enables a resource adapter to control the execution context of a `Work` instance that it has submitted to Payara Server for execution. The `Generic` work contract provides the mechanism for a resource adapter to augment the runtime context of a `Work` instance with

additional contextual information flown-in from the EIS. This contract enables a resource adapter to control, in a more flexible manner, the contexts in which the `Work` instances submitted by it are executed by the application server's `WorkManager`. **Security work context**: A standard contract that enables a resource adapter to establish security information while submitting a `Work` instance for execution to a `WorkManager` and while delivering messages-to-message endpoints residing in the Payara Server. This contract provides a mechanism to support the execution of a `Work` instance in the context of an established identity. It also supports the propagation of user information or `Principal` information from an EIS to a `MessageEndpoint` during message inflow. **Transaction context**: The transaction context contract between the resource adapter and the application server leverages the Generic Work Context mechanism by describing a standard `WorkContext`, the `TransactionContext`. It represents the standard interface a resource adapter can use to propagate transaction context information from the EIS to the application server.

- The Connector 2.0 specification moves the old Java Connectors Architecture specification (JCA) to Jakarta EE.
- The Connector 2.1 specification, as part of Jakarta EE 10 has introduced Java SE 11 as the minimum requirement to run Jakarta EE applications.

Connector Architecture for JMS and JDBC

In the Administration Console, connector, JMS, and JDBC resources are handled differently, but they use the same underlying Connector architecture.

In Payara Server, all communication to an EIS, whether to a message provider or an RDBMS, happens through the Connector architecture. To provide JMS infrastructure to clients, Payara Server uses the **Open Message Queue** (OpenMQ) software. To provide JDBC infrastructure to clients, Payara Server uses its own JDBC system resource adapters. Payara Server automatically makes these system resource adapters available to any client that requires them.

For more information about JMS in the Payara Server, see using-jakarta-messaging.

For more information about JDBC in the Payara Server, see using-the-jdbc-api-for-database-access.

Connector Configuration

Payara Server does not need to use `sun-ra.xml` deployment descriptor, which previous versions used (as with GlassFish), to store server-specific deployment information inside a Resource Adapter Archive (RAR) file. (However, the `sun-ra.xml` file is still supported for backward compatibility.) Instead, the information is stored in the server configuration.

As a result, you can create multiple connector connection pools for a connection definition in a functional resource adapter instance, and you can create multiple user-accessible connector resources (that is, registering a resource with a JNDI name) for a connector connection pool.

TIP	Dynamic changes can be made to connector connection pools and the connector resource properties without restarting Payara Server.

Advanced Connector Configuration Options

Thread Associations

Connectors can submit work instances to Payara Server for execution. By default, Payara Server services work requests for all connectors from its default thread pool. However, you can associate a specific user-created thread pool to service work requests from a connector. A thread pool can service work requests from multiple resource adapters.

To create a thread pool:

- In the Administration Console, select *Thread Pools* under the relevant configuration.
- Use the `asadmin create-threadpool` command. For details, see the "create-threadpool".

To associate a connector with a thread pool:

- In the Administration Console, open the Applications component and select Resource Adapter Configs. Specify the name of the thread pool in the Thread Pool ID field.
- Use the `--threadpoolid` option of the `asadmin create-resource-adapter-config` command. For details, see the "create-resource-adapter-config".

> **NOTE** If you create a resource adapter configuration for a connector module that is already deployed, the connector module deployment is restarted with the new configuration properties.

Security Maps

Create a security map for a connector connection pool to map an application principal or a user group to a backend EIS principal. The security map is usually used in situations where one or more EIS back end principals are used to execute operations (on the EIS) initiated by various principals or user groups in the application.

To create or update security maps for a connector connection pool:

- In the Administration Console, open the *Resources* component, select *Connectors*, select *Connector Connection Pools*, then select the corresponding connector connection pool, and select the _ Security Maps_ tab.
- Use the `asadmin create-connector-security-map` command. For details, see the""create-connector-security-map".

If a security map already exists for a connector connection pool, the new security map is appended to the previous one. The connector security map configuration supports the use of the wildcard asterisk (`*`) to indicate all users or all user groups.

When an application principal initiates a request to an EIS, Payara Server first checks for an exact match to a mapped back end EIS principal using the security map defined for the connector connection pool. If there is no exact match, Payara Server uses the wild card character specification, if any, to determine the mapped back end EIS principal.

Work Security Maps

A work security map for a resource adapter maps an EIS principal or group to an application principal or group. A work security map is useful in situations where one or more application principals execute operations initiated by principals or user groups in the EIS. A resource adapter can have multiple work security maps. A work security map can map either principals or groups, but not both.

To create a work security map, use the `asadmin create-connector-work-security-map` command. For details, see the "create-connector-work-security-map".

The work security map configuration supports the wildcard asterisk (`*`) character to indicate all users or all user groups. When an EIS principal sends a request to Payara Server, it will first check for an exact match to a mapped application principal using the work security map defined for the resource adapter.

If there is no exact match, Payara Server uses the wild card character specification, if any, to determine the application principal.

Overriding Configuration Properties

You can override the properties (`config-property` elements) specified in the `ra.xml` file of a resource adapter:

- In the Administration Console, open the *Resources* component and select *Resource Adapter Configs*. Create a new resource adapter configuration or select an existing one to edit. Then enter property names and values in the *Additional Properties* table.
- Use the `asadmin create-resource-adapter-config` command to create a configuration for a resource adapter. Use this command's `--property` option to specify a name-value pair for a resource adapter property. For details, see the "create-resource-adapter-config".

You can specify configuration properties either before or after resource adapter deployment. If you specify properties after deploying the resource adapter, the existing resource adapter is restarted with the new properties.

You can also use token replacement for overriding resource adapter configuration properties in individual server instances when the resource adapter is deployed to a cluster. For example, for a property called `inboundPort`, you can assign the value `${inboundPort}`. You can then assign a different value to this system property for each server instance. Changes to system properties take effect upon server restart.

Testing a Connector Connection Pool

You can test a connector connection pool for usability in one of these ways:

- In the Administration Console, open the *Resources* component, open the *Connectors* component, select *Connection Pools*, and select the connection pool you want to test. Then select the *Ping* button in the top right corner of the page.
- Use the `asadmin ping-connection-pool` command. For details, see the "ping-connection-pool".

Both these commands fail and display an error message unless they successfully connect to the connection pool.

You can also specify that a connection pool is automatically tested when created or reconfigured by setting the *Ping* attribute to `true` (the default is `false`) in one of the following ways:

- Enter a *Ping* value in the *Connector Connection Pools* page in the Administration Console.
- Specify the `--ping` option in the `asadmin create-connector-connection-pool` command. For more information, see the "create-connector-connection-pool".

Flushing a Connector Connection Pool

Flushing a connector connection pool recreates all the connections in the pool and brings the pool to the steady pool size without the need for reconfiguring the pool. Connection pool reconfiguration can result in application redeployment, which is a time-consuming operation. Flushing destroys existing connections, and any existing transactions are lost and must be retired.

You can flush a connector connection pool in one of these ways:

- In the Administration Console, open the *Resources* component, open the *Connectors* component, select *Connection Pools*, and select the connection pool you want to flush. Then select the *Flush* button in the top right corner of the page.
- Use the `asadmin flush-connection-pool` command. For details, see the "flush-connection-pool".

Handling Invalid Connections

If a resource adapter generates a `ConnectionErrorOccured` event, Payara Server considers the connection invalid and removes the connection from the connection pool. Typically, a resource adapter generates a `ConnectionErrorOccured` event when it finds a `ManagedConnection` object unusable. Reasons can be network failure with the EIS, EIS failure, fatal problems with the resource adapter, and so on.

If the `fail-all-connections` setting in the connection pool configuration is set to `true`, and a single connection fails, all connections are closed and recreated. If this setting is `false`, individual connections are recreated only when they are used. The default is `false`.

The `is-connection-validation-required` setting specifies whether connections have to be validated before being given to the application. If a resource's validation fails, it is destroyed, and a new resource is created and returned. The default value is `false`.

The `prefer-validate-over-recreate` property specifies that validating idle connections is preferable to closing them. This property has no effect on non-idle connections. If set to `true`, idle connections are validated during pool resizing, and only those found to be invalid are destroyed and recreated. If `false`, all idle connections are destroyed and recreated during pool resizing. The default value is `false`.

You can set the `fail-all-connections`, `is-connection-validation-required`, and `prefer-validate-over-recreate` configuration settings during creation of a connector connection pool. Or, you can use the `asadmin set` command to dynamically reconfigure a setting. For example:

```
asadmin set server.resources.connector-connection-pool.CCP1.fail-all-connections
="true"
asadmin set server.resources.connector-connection-pool.CCP1.is-connection-
validation-required="true"
asadmin set server.resources.connector-connection-pool.CCP1.property.prefer-
validate-over-recreate="true"
```

The interface ValidatingManagedConnectionFactory exposes the method `getInvalidConnections` to allow retrieval of the invalid connections. Payara Server checks if the resource adapter implements this interface, and if it does, invalid connections are removed when the connection pool is resized.

Setting the Shutdown Timeout

According to the Jakarta Connectors specification, while an application server shuts down, all resource adapters SHOULD be stopped. A resource adapter might hang during shutdown, since shutdown is typically a resource intensive operation. To avoid such a situation, you can set a timeout that aborts resource adapter shutdown if exceeded.

The default timeout is `30 seconds` per resource adapter module. To configure this timeout:

- In the Administration Console, select *Connector Service* under the relevant configuration and edit the *Shutdown Timeout* field.
- Use the following `asadmin set` command:

```
asadmin set server.connector-service.shutdown-timeout-in-seconds="num-secs"
```

WARNING	Payara Server deactivates all message-driven bean deployments before stopping a resource adapter.

Specifying the Class Loading Policy

Use the `class-loading-policy` setting to determine which resource adapters are accessible to applications. Allowed values are:

`derived`: Applications access resource adapters based on references in their deployment descriptors. These references can be `resource-ref`, `resource-env-`

`ref`, `resource-adapter-mid`, or equivalent annotations. **`global`**: All stand-alone resource adapters are available to all applications.

To configure this setting, use the `asadmin set` command. For example:

```
asadmin set server.connector-service.class-loading-policy="global"
```

Using Last Agent Optimization of Transactions

Transactions that involve multiple resources or multiple participant processes are distributed or global transactions. A global transaction can involve one non-XA resource if last agent optimization is enabled. Otherwise, ALL resources must be XA-aware.

For more information about transactions in Payara Server, see using-the-transaction-service.

The Jakarta Connectors specification requires that if a resource adapter supports `XATransaction`, the `ManagedConnection` created from that resource adapter must support BOTH distributed and local transactions. Therefore, even if a resource adapter supports `XATransaction`, you can configure its connector connection pools as non-XA or without transaction support for better performance.

NOTE	A non-XA resource adapter becomes the last agent in the transactions in which it participates.

The value of the connection pool configuration property `transaction-support` defaults to the value of the `transaction-support` property in the `ra.xml` file.

The connection pool configuration property can override the `ra.xml` file property if the transaction level in the connection pool configuration property is lower. If the value in the connection pool configuration property is higher, it is ignored.

Disabling Pooling for a Connection

To disable connection pooling, set the *Pooling* attribute to `false`. The default is `true`. You can enable or disable connection pooling in one of the following ways:

- Enter a *Pooling* value in the *Connector Connection Pools* page in the Administration Console.
- Specify the `--pooling` option in the `asadmin create-connector-connection-pool` command. For more information, see the "create-connector-connection-pool".

Using Application-Scoped Connectors

You can define an application-scoped connector or other resource for an enterprise application, web module, EJB module, connector module, or application client module by supplying a `payara-resources.xml` or `glassfish-resources.xml` deployment descriptor file. For details, see "application-scoped-resources" in the Payara Server Application Deployment section.

IMPORTANT	glassfish-resources.xml is deprecated and support will be removed in a future release. It is recommended to use payara-resources.xml instead.

Inbound Communication Support

The Jakarta Connectors specification defines the transaction and message inflow system contracts for achieving inbound connectivity from an EIS. The message inflow contract also serves as a standard message provider's pluggability contract, thereby allowing various message providers to seamlessly plug in their products with any application server that supports the message inflow contract.

In the inbound communication model, the EIS initiates all communication to an application. An application can be composed of enterprise beans (session, entity, or message-driven beans), which reside in an EJB container.

Incoming messages are received through a message endpoint, which is a message-driven bean. This message-driven bean asynchronously consumes messages from a message provider. An application can also synchronously send and receive messages directly using messaging style APIs.

A resource adapter supporting inbound communication provides an instance of an `ActivationSpec` JavaBean class for each supported message listener type. Each class contains a set of configurable properties that specify endpoint activation configuration information during message-driven bean deployment. The required `config-property` element in the `ra.xml` file provides a list of configuration property names required for each activation specification.

WARNING

An endpoint activation fails if the required property values are not specified. Values for the properties that are overridden in the message-driven bean's deployment descriptor are applied to the `ActivationSpec` JavaBean when the message-driven bean is deployed.

Administered objects can also be specified for a resource adapter, and these JavaBeans are specific to a messaging style or message provider. For example, some messaging styles may need applications to use special administered objects (such as `Queue` and `Topic` objects in JMS). Applications use these objects to send and synchronously receive messages using connection objects using messaging style APIs. For more information about administered objects, see using-jakarta-messaging.

Outbound Communication Support

The Jakarta Connectors specification defines the system contracts for achieving outbound connectivity from an EIS. A resource adapter supporting outbound communication provides an instance of a `ManagedConnectionFactory` JavaBean class. A `ManagedConnectionFactory` JavaBean represents outbound connectivity information to an EIS instance from an application.

Version 1.6 of the specification introduced a mechanism through which the transaction level of a `ManagedConnectionFactory` can be detected at runtime. During the configuration of a `ManagedConnectionFactory` in the Connector Connection Pools page in the Administration Console, the Administration Console can instantiate the `ManagedConnectionFactory` and show the level of transaction support.

The three levels are `no-tx`, `local-tx`, `xa-tx`. If a `ManagedConnectionFactory` returns `local-tx` as the level it can support, it is assumed that `xa-tx` is not supported, and the Administration Console shows only `no-tx` and `local-tx` as the available support levels.

Configuring a Message Driven Bean to Use a Resource Adapter

The Connectors specification's message inflow contract provides a generic mechanism to plug in a wide-range of message providers, including JMS, into a Java-EE-compatible application server. Message providers use a resource adapter and dispatch messages to message endpoints, which are implemented as message-driven beans.

The message-driven bean developer provides activation configuration information in the message-driven bean's `ejb-jar.xml` file. Configuration information includes messaging-style-specific configuration details, and possibly message-provider-specific details as well. The message-driven bean deployer uses this configuration information to set up the activation specification JavaBean. The activation configuration properties specified in `ejb-jar.xml` override configuration properties in the activation specification definition in the `ra.xml` file.

According to the Jakarta Enterprise Beans specification, the messaging-style-specific descriptor elements contained within the activation configuration element are not specified because they are specific to a messaging provider. In the following sample message-driven bean `ejb-jar.xml`, a message-driven bean has the following activation configuration property names: `destinationType`, `SubscriptionDurability`, and `MessageSelector`.

```
<!--  A sample MDB that listens to a JMS Topic -->
<!-- message-driven bean deployment descriptor -->
 <activation-config>
   <activation-config-property>
     <activation-config-property-name>
       destinationType
     </activation-config-property-name>
     <activation-config-property-value>
       jakarta.jms.Topic
     </activation-config-property-value>
  </activation-config-property>
  <activation-config-property>
```

```
      <activation-config-property-name>
        SubscriptionDurability
      </activation-config-property-name>
      <activation-config-property-value>
        Durable
      </activation-config-property-value>
   </activation-config-property>
   <activation-config-property>
      <activation-config-property-name>
        MessageSelector
      </activation-config-property-name>
      <activation-config-property-value>
        JMSType = 'car' AND color = 'blue'
      </activation-config-property-value>
   </activation-config-property>
  ...
  </activation-config>
```

When the message-driven bean is deployed, the value for the `resource-adapter-mid` element in the `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml` file is set to the resource adapter module name that delivers messages to the message endpoint (to the message-driven bean). In the following example, the `jmsra` JMS resource adapter, which is the bundled resource adapter for the Message Queue message provider, is specified as the resource adapter module identifier for the `SampleMDB` bean.

```
<payara-ejb-jar>
<enterprise-beans>
    <unique-id>1</unique-id>
    <ejb>
       <ejb-name>SampleMDB</ejb-name>
       <jndi-name>SampleQueue</jndi-name>
    <!-- JNDI name of the destination from which messages would be
         delivered from MDB needs to listen to -->
    ...
    <mdb-resource-adapter>
       <resource-adapter-mid>jmsra</resource-adapter-mid>
       <!-- Resource Adapter Module Id that would deliver messages to
            this message endpoint -->
       </mdb-resource-adapter>
    ...
 </ejb>
 ...
</enterprise-beans>
...
</payara-ejb-jar>
```

When the message-driven bean is deployed, Payara Server uses the `resourceadapter-mid` setting to associate the resource adapter with a message endpoint through the message inflow contract. This message inflow contract with the

server gives the resource adapter a handle to the `MessageEndpointFactory` and the `ActivationSpec` JavaBean, and the adapter uses this handle to deliver messages to the message endpoint instances (which are created by the `MessageEndpointFactory`).

NOTE

When a message-driven bean first created for use on Payara Server is deployed, the connector runtime transparently transforms the previous deployment style to the current connector-based deployment style. If the deployer specifies neither a `resource-adapter-mid` element nor the Message Queue resource adapter's activation configuration properties, the Connector runtime maps the message-driven bean to the `jmsra` system resource adapter and converts the JMS-specific configuration to the Message Queue resource adapter's activation configuration properties.

Developing Osgi-Enabled Jakarta EE Applications

This chapter describes the features and interfaces that Payara Server provides to develop OSGi-enabled Enterprise applications.

Overview of Osgi Application and Payara Server

Payara Server is a certified Jakarta EE implementation, so it provides the latest Jakarta EE APIs and frameworks. It is built using OSGi technology, and includes as its OSGi module management subsystem the Apache Felix OSGi framework, which is a fully-compliant implementation of the OSGi Service Platform R7 specification. Payara Server supports deployment of OSGi-based applications using this framework.

OSGi applications can make use of core as well as enterprise OSGi features. Payara Server makes available many of its Jakarta EE platform services, such as the transaction service, HTTP service, JDBC Service and JMS, as OSGi services. It also enables use of Jakarta EE programming model in OSGi applications, so enterprise Jakarta application developers can continue to leverage their existing skills in OSGi-based applications. See Benefits of Using OSGi in Enterprise Jakarta Applications for

more information.

OSGi applications are deployed as one or more OSGi bundles, and the Payara Server deployment and administration infrastructure enables you to deploy and manage your OSGi bundles. This chapter classifies OSGi bundles into two categories based on the features they use:

Plain OSGi Application Bundles: Bundles that do not contain any Jakarta EE components. See Developing Plain OSGi Bundles.

Hybrid Application Bundles: Bundles that are an OSGi bundle as wells as a Jakarta EE module. At runtime, such modules have both an OSGi bundle context and a Jakarta EE context. Payara Server supports the following hybrid application bundles: * Web Application Bundles (or WABs) , see Developing Web Application Bundles. * EJB Application Bundles, see Developing EJB Application Bundles.

Benefits of Using Osgi in Enterprise Jakarta Applications

Enterprise applications typically need transactional, secured access to data stores, messaging systems and other such enterprise information systems, and have to cater to a wide variety of clients such as web browsers and desktop applications, and so on. Jakarta EE makes development of such applications easier with a rich set of APIs and frameworks. It also provides a scalable, reliable and easy to administer runtime to host such applications.

The OSGi platform complements these features with modularity. It enables applications to be separated into smaller, reusable modules with a well-defined and robust dependency specification.

A module explicitly specifies its capabilities and requirements. This explicit dependency specification encourages developers to visualize dependencies among their modules and help them make their modules highly cohesive and less coupled. The OSGi module system is dynamic: It allows modules to be added and removed at runtime.

OSGi has very good support for versioning too: It supports **package versioning** as well

as **module versioning**. In fact, it allows multiple versions of the same package to coexist in the same runtime, thus allowing greater flexibility to deployers. The service layer of the OSGi platform encourages a more service-oriented approach to build complex systems.

The service-oriented approach and dynamic module system used together allow a system to be more agile during development as well as in production. It makes them better suited to run in a Platform-as-a-Service (PaaS) environment.

With Payara Server, you do not have to chose one of the two platforms. A hybrid approach like OSGi enabling your Jakarta EE applications allows new capabilities to applications hitherto unavailable to applications built using just one of the two platforms.

Developing Osgi Application Bundles for Payara Server

Payara Server enables interaction between OSGi components and Jakarta EE components.

OSGi services managed by the OSGi framework can invoke Jakarta EE components managed by the Jakarta EE container and vice versa. For example, developers can declaratively export EJBs as OSGi services without having to write any OSGi code. This allows any plain OSGi component, which is running without the Jakarta EE context, to discover the EJB and invoke it.

Similarly, Jakarta EE components can locate OSGi services provided by plain OSGi bundles and use them as well. Payara Server extends the Jakarta EE Context and Dependency Injection (CDI) framework to make it easier for Jakarta EE components to consume dynamic OSGi services in a type-safe manner.

Developing Plain Osgi Bundles

Jakarta EE components (like an EJB or Servlet) can look up Jakarta EE platform services using JNDI names in the associated Jakarta EE naming context. Such code can rely on the Jakarta EE container to inject the required services as well. Unfortunately, neither of them works when the code runs outside a Jakarta EE context.

An example of such code is the `BundleActivator` of an OSGi bundle. For such code to access Jakarta EE platform services, Payara Server makes key services and resources of the underlying Jakarta EE platform available as OSGi services.

Thus, an OSGi bundle deployed in Payara Server can access these services using OSGi Service look-up APIs or by using a white board pattern. The following Jakarta EE services are available as OSGi services:

- HTTP Service
- Transaction Service
- JDBC Data Source Service
- JMS Resource Service

HTTP Service

The Payara Server web container is made available as a service for OSGi users who do not use OSGi Web Application Bundles (WABs).

This service is made available using the standard OSGi/HTTP service specification, which is a lightweight API that predates the concept of a web application as we know it today.

This simple API allows users to register servlets and static resources dynamically and draw a boundary around them in the form of a `HttpContext`.

This simple API can be used to build feature-rich web application, such as the Felix Web Console for example.

The Payara Server web container has one or more **virtual servers**. A virtual server has one or more web application deployed in it. Each web application has a distinct context path.

Each virtual server has a set of HTTP listeners. Each HTTP listener listens on a particular port. When multiple virtual servers are present, one of them is treated as the default virtual server. Every virtual server comes configured with a default web

application. The default web application is used to serve static content from the `docroot` directory of a Payara Server domain. This default web application uses `/` as the context path. A web application contains static and dynamic resources. Each virtual server is mapped to an `org.osgi.services.http.HttpService` instance. When there are multiple virtual servers present, there will be multiple occurrences of `HttpService` registered in the service registry.

In order to distinguish one service from another, each service is registered with a service property named `VirtualServer`, whose value is the name of the virtual server.

The service corresponding to default virtual server has the highest ranking, so when looking up a service of type `HttpService` without any additional criteria returns the `HttpService` corresponding to the default virtual server. In a typical installation, the default virtual server is configured to listen on port `8080` for the HTTP protocol and port `8181` for the HTTPS protocol.

The context path `/` is reserved for the default web application. Every resource and servlet registered using the `registerResource()` and `registerServlet()` methods of `HttpService` are made available under a special context path named `/osgi` in the virtual server. The `/osgi` context path can be changed to some other value by setting an appropriate value in the OSGi configuration property or in a system property called `org.glassfish.osgihttp.ContextPath`.

For example, a `HelloWorldServlet` component will be available at `http://localhost:8080/osgi/helloworld` when the following code is executed:

```
HttpService httpService = getHttpService(); // Obtain HttpService
httpService.registerServlet(httpService.registerServlet("/helloworld", new
HelloWorldServlet(), null, ctx);
```

Transaction Service

The Jakarta Transaction API (JTA) defines three interfaces to interact with the transaction management system: `UserTransaction`, `TransactionManager`, and `TransactionSynchronizationRegistry`. They all belong to the

`jakarta.transaction` package. `TransactionManager`and `TransactionSynchronizationRegistry` are intended for system level code, such as a persistence provider. Whereas, `UserTransaction` is the entity that you should use to control user-managed transactions.

All the objects of the underlying JTA layer are made available in the OSGi service registry using the following service interfaces:

- `jakarta.transaction.UserTransaction`
- `jakarta.transaction.TransactionManager`
- `jakarta.transaction.TransactionSynchronisationRegistry`

There is no additional service property associated with them. Although `UserTransaction` appears to be a singleton, in reality any call to it gets rerouted to the actual transaction associated with the calling thread. Code that runs in the context of a Jakarta EE component typically gets a handle on `UserTransaction` by doing a JNDI lookup in the component naming context or by using injection, as shown here:

```
var utx = (UserTransaction)(new InitialContext().lookup(
"java:comp/UserTransaction"));
```

or

```
@Resource
UserTransaction utx;
```

When certain code (such as an OSGi Bundle Activator), which does not have a Jakarta EE component context, wants to get hold of `UserTransaction`, or any of the other JTA artifacts, then they can look it up in the service registry. Here is an example of such code:

```
BundleContext context; // An OSGi context must be present
var txRef = context.getServiceReference(UserTransaction.class.getName());
var utx = (UserTransaction)context.getService(txRef);
```

JDBC Data Source Service

Any JDBC data source created in Payara Server is automatically made available as an OSGi Service; therefore, OSGi bundles can track availability of JDBC data sources using the `ServiceTracking` facility of the OSGi platform.

The life of the OSGi service matches that of the underlying data source created in Payara Server.

Payara Server registers each JDBC data source as an OSGi service with `objectClass = "javax.sql.DataSource"` and a service property called `jndi-name`, which is set to the JNDI name of the data source.

Here is a code sample that looks up a data source service:

```
@Inject
@OSGiService(filter = "(jndi-name=jdbc/MyDS)")
private DataSource ds;
```

JMS Resource Service

Like JDBC data sources, JMS administered objects, such as destinations and connection factories, are also automatically made available as OSGi services.

Their service mappings are as follows.

JMS Object	Service Interface	Service Properties	Comments
JMS Queue destination	`jakarta.jms.Queue`	`jndi-name`	`jndi-name` is set to the JNDI name of the queue
JMS Topic destination	`jakarta.jms.Topic`	`jndi-name`	`jndi-name` is set to the JNDI name of the topic

JMS Object	Service Interface	Service Properties	Comments
JMS connection factory	• `jakarta.jms.QueueConnectionFactory` • `jakarta.jms.TopicConnectionFactory` • `jakarta.jms.ConnectionFactory`	`jndi-name`	`jndi-name` is set to the JNDI name of the topic. The actual service interface depends on which type of connection factory was created.

Developing Web Application Bundles

When a web application is packaged and deployed as an OSGi bundle, it is called a **Web Application Bundle (WAB)**. WAB support is based on the OSGi Web Application specification, which is part of the OSGi Service Platform, Enterprise Specification, Release 7. A WAB is packaged as an OSGi bundle, so all the OSGi packaging rules apply to WAB packaging.

When a WAB is not packaged like a WAR, the OSGi Web Container of Payara Server maps the WAB to the hierarchical structure of web application using the following rules:

- The root of the WAB corresponds to the `docroot` of the web application.
- Every JAR in the `Bundle-ClassPath` attribute of the WAB is treated like a JAR located in `WEB-INF/lib/.`
- Every directory except "." in the `Bundle-ClassPath` attribute of the WAB is treated like the `WEB-INF/classes/.` directory.
- A `Bundle-ClassPath` entry of type "." is treated as if the entire WAB is a JAR located in `WEB-INF/lib/.`
- `Bundle-ClassPath` includes the `Bundle-ClassPath` entries of any attached fragment bundles.

TIP

We strongly recommend that you package the WAB exactly like a WAR, with only additional OSGi metadata to prevent triggering any conflicts with the rules mentioned above.

WAB Metadata

In addition to the standard OSGi metadata, the main attributes of the `META-INF/MANIFEST.MF` of the WAB must have an additional attribute called `Web-ContextPath`.

The `Web-ContextPath` attribute specifies the value of the context path of the web application. Since the root of a WAB is mapped to the `docroot` of the web application, it should not be used in the `Bundle-ClassPath` attribute.

Moreover, `WEB-INF/classes/` should be specified ahead of `WEB-INF/lib/` in the `Bundle-ClassPath` in order to be compliant with the search order used for traditional WAR files.

Assuming the WAB is structured as follows:

```
foo.war/
   index.html
   foo.jsp
   WEB-INF/classes/
      foo/BarServlet.class
   WEB-INF/lib/lib1.jar
   WEB-INF/lib/lib2.jar
```

Then the OSGi metadata for the WAB as specified in `META-INF/MANIFEST.MF` of the WAB would appear as follows:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-SymbolicName: com.acme.foo
Bundle-Version: 1.0
Bundle-Name: Foo Web Application Bundle Version 1.0
Import-Package: jakarta.servlet; jakarta.servlet.http, version=[3.0, 4.0, 5.0)
Bundle-ClassPath: WEB-INF/classes, WEB-INF/lib/lib1.jar, WEB-INF/lib/lib2.jar
Web-ContextPath: /foo
```

How Wabs Consume Osgi Services

Since a WAB has a valid `Bundle-Context`, it can consume OSGi services. Although you are free to use any OSGi API to locate OSGi services, Payara Server makes it easy for WAB users to use OSGi services by virtue of extending the Context and Dependency Injection (CDI) framework. Here's an example of the injection of an OSGi Service into a Servlet:

```
@WebServlet
public class MyServlet extends HttpServlet {

    @Inject
    @OSGiService(dynamic=true)
    FooService fooService;
}
```

Osgi CDI Extension for WAB

Payara Server includes a CDI extension that enables web applications, such as servlets, that are part of WABs to express a type-safe dependency on an OSGi service using CDI APIs.

An OSGi service can be provided by any OSGi bundle without any knowledge of Jakarta EE/CDI, and they are allowed to be injected transparently in a type-safe manner into a web application.

A custom CDI Qualifier, `@org.glassfish.osgicdi.OSGiService`, is used by the component to represent dependency on an OSGi service. The qualifier has additional metadata to customize the service discovery and injection behavior.

The following `@OsgiService` attributes are currently available:

- `serviceCriteria` — An LDAP filter query used for service selection in the OSGi service registry.
- `waitTimeout` — Waits the specified duration for a service that matches the criteria specified to appear in the OSGi service registry.
- `dynamic` — Dynamically obtain a service reference (`true`/`false`).

NOTE	Since OSGi services are dynamic, they may not match the life cycle of the application component that has injected a reference to the service. Through the `dynamic` attribute, you can indicate that a service reference can be obtained dynamically or not. For stateless or idempotent services, a dynamic reference to a service implementation would be useful. The container then injects a proxy to the service and dynamically switches to an available implementation when the current service reference is invalid.

Example

In this example, Bundle B0 defines a service contract called `com.acme.Foo` and exports the `com.acme` package for use by other bundles.

Bundle B1 in turn provides a service implementation, `FooImpl`, of the `com.acme.Foo` interface. It then registers the service FooImpl service with the OSGi service registry with `com.acme.Foo` as the service interface.

Bundle B2 is a hybrid application bundle that imports the `com.acme` package. It has a component called `BarServlet` that expresses a dependency to `com.acme.Foo` by adding a field/setter method and qualifies that injection point with a `@OsgiService` annotation.

For instance, `BarServlet` looks like this:

```
@Servlet
public class BarServlet extends HttpServlet{

    @Inject
    @OSGiService(dynamic=true)
    private com.acme.Foo f;
}
```

EJB Application Bundles

Another type of hybrid application bundle is the EJB Application Bundle. When an EJB Jar is packaged with additional OSGi metadata and deployed as an OSGi bundle it is called an EJB Application Bundle.

Payara Server supports only packaging the OSGi bundle as a simple JAR file with the required OSGi metadata, just as you would package an `ejb-jar` file.

Required EJB Metadata

An EJB Application Bundle must have a manifest metadata attribute called `Export-EJB` in order to be considered as an EJB Bundle. Here's an example of an EJB Application Bundle with this metadata attribute:

```
myEjb.jar/
   com/acme/Foo
   com/acme/impl/FooEJB
   META-INF/MANIFEST.MF
```

And here's how its Manifest entries look like:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-SymbolicName: com.acme.foo EJB bundle
Bundle-Version: 1.0.0.BETA
Bundle-Name: com.acme.foo EJB bundle version 1.0.0.BETA
Export-EJB: ALL
Export-Package: com.acme; version=1.0
Import-Package: jakarta.ejb; version=[3.0, 4.0), com.acme; version=[1.0, 1.1)
```

How EJB Bundles Consume Osgi Services

Since an EJB has a valid Bundle-Context, it can consume OSGi services. Although you are free to use any OSGi API to locate OSGi services, Payara Server makes it easy to use OSGi services by virtue of extending the Context and Dependency Injection (CDI) framework.

Here's an example of injection of an OSGi Service into a stateless bean:

```
@Stateless
public class MyEJB {

    @Inject
    @OSGiService(dynamic=true)
    Foo foo;
}
```

Using the Osgi CDI Extension with EJB Bundles

Payara Server includes a CDI extension that enables EJB application bundles to express a type-safe dependency on an OSGi Service using CDI APIs.

An OSGi service can be provided by any OSGi bundle without any knowledge of Jakarta EE/CDI, and they are allowed to be injected transparently in a type-safe manner into an EJB bundle.

A custom CDI Qualifier, `@org.glassfish.osgicdi.OSGiService`, is used by the component to represent dependency on an OSGi service. The qualifier has additional metadata to customize the service discovery and injection behavior.

The following `@OsgiService` attributes are currently available:

- `serviceCriteria` — An LDAP filter query used for service selection.
- `waitTimeout` — Waits for specified duration for a service to appear in the OSGi service registry.
- `dynamic` — Dynamically obtain a service reference (`true`/`false`).

Deploying Osgi Bundles in Payara Server

For instruction on deploying OSGi bundle, see "osgi-bundle-deployment-guidelines" in the Payara Server Application Deployment section.

Using the JDBC API for Database Access

This chapter describes how to use the Java Database Connectivity (JDBC) API for database access with Payara Server.

This chapter also provides high level JDBC implementation instructions for servlets and EJB components deployed in Payara Server.

IMPORTANT	Payara Server does not support connection pooling or transactions for an application's database access if it does not use standard Jakarta EE `DataSource` objects.

Statements

Using an Initialization Statement

You can specify a SQL statement that executes each time a **physical** connection to the database is created (not reused) from a JDBC connection pool. This is useful for setting request or session specific properties and is suited for homogeneous requests in a single application.

Set the **Init SQL** attribute of the JDBC connection pool to the SQL string to be executed in one of the following ways:

- Enter an *Init SQL value* in the *Edit Connection Pool Advanced Attributes* page in the Administration Console.
- Specify the `--initsql` option in the `asadmin create-jdbc-connection-pool` command.
- Specify the `init-sql` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.init-sql="sql-string"
```

Setting a Statement Timeout

An abnormally long-running SQL query executed by an application may leave it in a hanging state unless a timeout is explicitly set on the statement. Setting a statement timeout guarantees that all queries automatically time out if not completed within the specified period.

When statements are created, the `queryTimeout` is set according to the statement timeout setting. This works only when the underlying JDBC driver supports `queryTimeout` for `Statement`, `PreparedStatement`, `CallableStatement`, and `ResultSet`.

You can specify a statement timeout in the following ways:

- Enter a *Statement Timeout* value in the *Edit Connection Pool Advance Attributes* page in the Administration Console.

- Specify the `--statementtimeout` option in the `asadmin create-jdbc-connection-pool` command.

Statement Leak Detection and Leaked Statement Reclamation

If SQL statements are not closed by an application after use, it is possible for the application to run out of cursors. Enabling statement leak detection causes statements to be considered as leaked if they are not closed within a specified period. Additionally, leaked statements can be reclaimed automatically.

To enable statement leak detection, set *Statement Leak Timeout In Seconds* for the JDBC connection pool to a positive, nonzero value in one of the following ways:

- Specify the `--statementleaktimeout` option in the `create-jdbc-connection-pool` subcommand. For more information, see "`create-jdbc-connection-pool`".
- Specify the `statement-leak-timeout-in-seconds` option in the `set` subcommand. For example:

```
asadmin set resources.jdbc-connection-pool.pool-name.statement-leak-timeout-in-seconds=300
```

When selecting a value for *Statement Leak Timeout In Seconds*, make sure that:

- It is less than the *Connection Leak Timeout*; otherwise, the connection could be closed before the statement leak is recognized.
- It is greater than the *Statement Timeout*; otherwise, a long-running query could be mistaken as a statement leak.

After enabling statement leak detection, enable leaked statement reclamation by setting the *Reclaim Leaked Statements* setting for the JDBC connection pool to a `true` value in one of the following ways:

- Specify the `--statementleakreclaim=true` option in the `create-jdbc-connection-pool` subcommand. For more information, see "`create-jdbc-`

`connection-pool`".

- Specify the `statement-leak-reclaim` option in the `set` subcommand. For example:

```
asadmin set resources.jdbc-connection-pool.pool-name.statement-leak-reclaim=true
```

Statement Caching

Statement caching stores statements, prepared statements, and callable statements that are executed repeatedly by applications in a cache, thereby improving performance. Instead of the statement being prepared each time, the cache is searched for an existing match. The overhead of parsing and creating new statements each time is eliminated.

IMPORTANT	Statement caching is usually a feature of the JDBC driver. Payara Server provides caching for drivers that do not support caching, so consider using this feature after studying whether your JDBC driver supports statement caching.

To enable this feature, set the *Statement Cache Size* for the JDBC connection pool in one of the following ways:

- Enter a *Statement Cache Size* value in the *Edit Connection Pool Advanced Attributes* page in the Administration Console.
- Specify the `--statementcachesize` option in the `asadmin create-jdbc-connection-pool` command.
- Specify the `statement-cache-size` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.statement-cache-
size=10
```

NOTE	By default, this attribute is set to zero and the statement caching is

> turned off.

To enable statement caching, you can set any positive nonzero value. The built-in cache eviction strategy is **LRU-based** (Least Recently Used).

When a connection pool is flushed, the connections in the statement cache are recreated.

SQL Trace Listeners

Payara Server provides support for custom SQL Trace Listeners. A **SQL Trace Listener** is registered against a data source and is called after each method call made on the JDBC connection pool.

SQL Trace Listeners allows developers to track all calls to the database and can be used to develop custom auditing, error handling or monitoring components. SQL Trace Listeners can be enabled globally on a data source if the class that implements it is on the server's *classpath*, or can be enabled on application specific data sources by including them in the application's WAR or EAR.

SQL Trace Listener Interface

The SQL Trace Listener interface is shown below:

```
public interface SQLTraceListener {
    /**
     * Notify listeners with SQL trace information.
     * @param record SQLTraceRecord that has information related
     * to the SQL operation
     */
    public void sqlTrace(SQLTraceRecord record);

}
```

To write a custom trace listener you need to implement the interface and override the `sqlTrace` method. The `SQLTraceRecord` object contains information about the executed call to the JDBC connection pool. The following is the list of properties that can be queried using their respective **getter** methods:

```
/**
```

```
 * Thread ID from which SQL statement originated.
 */
private long threadID;

/**
 * Thread Name from which SQL statement originated.
 */
private String threadName;

/**
 * Pool Name in which the SQL statement is executed.
 */
private String poolName;

/**
 * Type of SQL query. Could be PreparedStatement, CallableStatement or
 * other object types.
 */
private String className;

/**
 * Method that executed the query.
 */
private String methodName;

/**
 * Time of execution of query.
 */
private long timeStamp;

/**
 * Parameters of the method that executed the SQL query. Includes information
 * like SQL query, arguments and so on.
 */
private Object[] params;
```

SQL Trace Listener Example

The following code fragment illustrates an example SQL trace listener that just logs the executed call to the server's log file:

```
public class SQLTraceLogger implements SQLTraceListener {

    private static Logger _logger = initLogger();

    private static Logger initLogger() {
        _logger = LogDomains.getLogger(SQLTraceLogger.class, LogDomains
.SQL_TRACE_LOGGER);
        return _logger;
    }

    public SQLTraceLogger() {
    }

    public void sqlTrace(SQLTraceRecord record) {
        _logger.log(Level.FINE, record.toString());
    }
```

```
}
```

SQL Trace Listener Logger

Payara Server provides an SQL tracing logger to log the SQL operations in the form of `SQLTraceRecord` objects in the `server.log` file. The module name under which the SQL operation is logged is `javax.enterprise.resource.sqltrace`.

SQL traces are logged as `FINE` messages along with the module name to enable easy filtering of the SQL logs. A sample SQL trace record looks like this:

```
[#|2009-11-27T15:46:52.202+0530|FINE|Payara Server
7.2.0|jakarta.enterprise.resource.sqltrace.com.sun.gjc.util
|_ThreadID=29;_ThreadName=Thread-
1;ClassName=com.sun.gjc.util.SQLTraceLogger;MethodName=sqlTrace;
|ThreadID=77 | ThreadName=p: thread-pool-1; w: 6 | TimeStamp=1259317012202
| ClassName=com.sun.gjc.spi.jdbc40.PreparedStatementWrapper40 |
MethodName=executeUpdate
| arg[0]=insert into table1(colName) values(100) | arg[1]=columnNames | |#]
```

This trace shows that an `executeUpdate(String sql, String columnNames)` operation is being done.

When SQL statement tracing is enabled and JDBC connection pool monitoring is enabled, Payara Server maintains a tracing cache of recent queries and their frequency of use. The following JDBC connection pool properties can be configured to control this cache and the monitoring statistics available from it:

- **`time-to-keep-queries-in-minutes`**: Specifies how long in minutes to keep a query in the tracing cache, tracking its frequency of use. The default value is `5` minutes.
- **`number-of-top-queries-to-report`**: Specifies how many of the most used queries, in frequency order, are listed the monitoring statistic. The default value is `10` queries.

Set these parameters in one of the following ways:

- Add them as custom properties in the *Edit JDBC Connection Pool Properties* page in the Administration Console.

- Specify them using the `--property` option in the `create-jdbc-connection-pool` subcommand. For more information, see "`create-jdbc-connection-pool`".
- Set them using the `set` subcommand. For example:

```
asadmin set resources.jdbc-connection-pool.pool-name.property.time-to-keep-queries-in-minutes=10
```

Configuring SQL Trace Listeners

To configure the listeners used by a connection pool set the *SQL Trace Listeners* attribute to a comma-separated list of trace listener implementation classes in one of the following ways:

- Enter an *SQL Trace Listeners* value in the *Edit Connection Pool Advanced Attributes* page in the Administration Console.
- Specify the `--sqltracelisteners` option in the `asadmin create-jdbc-connection-pool` command.
- Specify the `sql-trace-listeners` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.sql-trace-listeners=listeners
```

To make custom implementations of this interface available to Payara Server, place the implementation classes in `as-install/lib`.

More information on how to configure SQL Trace listeners can be read in the "JDBC Connection Pool" configuration of Payara Server's administration section.

Connections

Transparent Pool Reconfiguration

When the properties or attributes of a JDBC connection pool are changed, the connection pool is destroyed and re-created. Normally, applications using the connection pool must be redeployed as a consequence. This restriction can be avoided

by enabling transparent JDBC connection pool reconfiguration. When this feature is enabled, applications do not need to be redeployed. Instead, requests for a new connections are blocked until the reconfiguration operation completes.

Connection requests from any "in-flight" transactions are served using the old pool configuration to complete the transaction. Then, connections are created using the pool's new configuration, and any blocked connection requests are served with connections from the re-created pool.

To enable transparent JDBC connection pool reconfiguration, set the `dynamic-reconfiguration-wait-timeout-in-seconds` property of the JDBC connection pool to a positive, nonzero value in one of the following ways:

- Add it as a property in the *Edit JDBC Connection Pool Properties* page in the Administration Console.
- Specify it using the `--property` option in the `create-jdbc-connection-pool` subcommand. For more information, see "`create-jdbc-connection-pool`".
- Set it using the `set` subcommand. For example:

```
asadmin set resources.jdbc-connection-pool.pool-name.property.dynamic-
reconfiguration-wait-timeout-in-seconds=15
```

This property specifies the time in seconds to wait for in-use connections to close and in-flight transactions to complete. Any connections in use or transaction in flight past this time must be retried.

Disabling Pooling

To disable connection pooling, set the *Pooling* attribute to false. The default is `true`. You can enable or disable connection pooling in one of the following ways:

- Enter a Pooling value in the *Edit Connection Pool Advanced Attributes* page in the Administration Console.
- Specify the `--pooling` option in the `asadmin create-jdbc-connection-pool` command.

- Specify the `pooling` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.H2Pool.pooling=false
```

TIP The `pooling` option and the system property `com.sun.enterprise.connectors.SwitchoffACCConnectionPooling`, which turns off connection pooling in the Application Client Container, do not affect each other.

An exception is thrown if `associate-with-thread` is set to `true` and pooling is disabled. An exception is thrown if you attempt to flush a connection pool when pooling is disabled. A warning is logged if the following attributes are used, because they are useful only in a pooled environment:

- `connection-validation`
- `validate-atmost-once-period`
- `match-connections`
- `max-connection-usage`
- `idle-timeout`

Associating Connections with Threads

To associate connections with a thread, set the *Associate With Thread* attribute to `true`. The default is `false`. A `true` setting allows connections to be saved as `ThreadLocal` in the calling thread.

IMPORTANT Connections get reclaimed only when the calling thread dies or when the calling thread is not in use and the pool has run out of connections.

If the setting is `false`, the thread must obtain a connection from the pool each time the thread requires a connection.

The *Associate With Thread* attribute associates connections with a thread such that

when the same thread is in need of connections, it can reuse the connections already associated with that thread. In this case, the overhead of getting connections from the pool is avoided.

However, when this value is set to `true`, you should verify that the value of the *Max Pool Size* attribute is comparable to the *Max Thread Pool Size* attribute of the thread pool. If the *Max Thread Pool Size* value is much higher than the *Max Pool Size* value, a lot of time is spent associating connections with a new thread after dissociating them from an older one. Use this attribute in cases where the thread pool should reuse connections to avoid this overhead.

You can set the Associate With Thread attribute in the following ways:

- Enter an *Associate With Thread* value in the *Edit Connection Pool Advanced* Attributes page in the Administration Console.
- Specify the `--associatewiththread` option in the `asadmin create-jdbc-connection-pool` command.
- Specify the `associate-with-thread` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.associate-with-thread=true
```

Custom Connection Validation

You can specify a custom implementation for *Connection Validation* that is faster or optimized for a specific database.

Set the *Validation Method* attribute to the value `custom-validation`.

Payara Server provides a public interface, `org.glassfish.api.jdbc.ConnectionValidation`, which you can implement to plug in your custom implementation.

The *Validation Classname* attribute, specifies the fully qualified name of the class that

implements the `ConnectionValidation` interface.

The *Validation Classname* attribute is required if *Connection Validation* is enabled and *Validation Method* is set to *Custom Validation*.

To enable this feature, set *Connection Validation*, *Validation Method*, and *Validation Classname* for the JDBC connection pool in one of the following ways:

- Enter *Connection Validation*, *Validation Method*, and *Validation Classname* values in the *Edit Connection Pool Advanced Attributes* page in the Administration Console.

  You can select from among validation class names for common databases in the *Validation Classname* field.

- Specify the `--isconnectionvalidatereq`, `--validationmethod`, and `--validationclassname` options in the `asadmin create-jdbc-connection-pool` command.
- Specify the `is-connection-validation-required`, `connection-validation-method`, and `validation-classname` options in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.MyPool.is-connection-
validation-required=true
asadmin set domain1.resources.jdbc-connection-pool.MyPool.connection-validation-
method=custom-validation
asadmin set domain1.resources.jdbc-connection-pool.MyPool.validation-classname
=impl-class
```

By default, optimized validation mechanisms are provided for DB2, H2, Apache Derby, MSSQL, MySQL, Oracle, PostgreSQL and Sybase databases.

Additionally, for JDBC 4.0 compliant database drivers, a validation mechanism is provided that uses the `Connection.isValid(0)` implementation.

Sharing Connections

When multiple connections acquired by an application use the same JDBC resource, the connection pool provides connection sharing within the same transaction scope.

For example, suppose **Bean A** starts a transaction and obtains a connection, then calls a method in **Bean B**. If **Bean B** acquires a connection to the same JDBC resource with the same sign-on information, and if **Bean A** completes the transaction, the connection can be shared.

NOTE

Connections obtained through a resource are shared only if the resource reference declared by the Jakarta EE component allows it to be shareable. This is specified in a component's deployment descriptor by setting the `res-sharing-scope` element to `Shareable` for the particular resource reference.
To turn off connection sharing, set `res-sharing-scope` to `Unshareable`.

For general information about connections and JDBC URLs, see "administering-database-connectivity" in the Payara Server General Administration section.

Marking Bad Connections

The default `DataSource` implementation in Payara Server provides a `markConnectionAsBad` method. A marked bad connection is removed from its connection pool when it is closed and is not returned to the pool. The method signature is as follows:

```
public class Datasource{

    public void markConnectionAsBad(java.sql.Connection connection){
    }
}
```

For example:

```
public class ApplicationComponent{

    public void operation(){
        var ds = (com.sun.appserv.jdbc.DataSource)context.lookup
("jdbc/myDataSource");
        try(var con = ds.getConnection(); //Connection will be destroyed during
close.
            var stmt = con.createStatement()){
           stmt.executeUpdate("Update");
```

```
            }
            catch (BadConnectionException e){
               ds.markConnectionAsBad(con);
            }
        }
    }
```

Handling Invalid Connections

If a `ConnectionErrorOccured` event occurs, Payara Server considers the connection invalid and removes the connection from the connection pool. Typically, a JDBC driver generates a `ConnectionErrorOccured` event when it finds a `ManagedConnection` object unusable. Reasons can be database failure, network failure with the database, fatal problems with the connection pool, and so on.

If the `fail-all-connections` setting in the connection pool configuration is set to `true`, and a single connection fails, all connections are closed and recreated. If this setting is `false`, individual connections are recreated only when they are used. The default is `false`.

The `is-connection-validation-required` setting specifies whether connections have to be validated before being given to the application. If a resource's validation fails, it is destroyed, and a new resource is created and returned. The default is `false`.

The `prefer-validate-over-recreate` property specifies that validating idle connections is preferable to closing them. This property has no effect on non-idle connections. If set to `true`, idle connections are validated during pool resizing, and only those found to be invalid are destroyed and recreated.

If `false`, all idle connections are destroyed and recreated during pool resizing. The default is `false`.

You can set the `fail-all-connections`, `is-connection-validation-required`, and `prefer-validate-over-recreate` configuration settings during creation of a JDBC connection pool. Or, you can use the `asadmin set` command to dynamically reconfigure these settings. For example:

```
asadmin set server.resources.jdbc-connection-pool.MyPool.fail-all-connections
```

```
="true"
asadmin set server.resources.jdbc-connection-pool.MyPool.is-connection-validation-
required="true"
asadmin set server.resources.jdbc-connection-pool.MyPool.property.prefer-validate-
over-recreate="true"
```

NOTE

The interface `ValidatingManagedConnectionFactory` exposes the method `getInvalidConnections` to allow retrieval of the invalid connections.
Payara Server checks if the JDBC driver implements this interface, and if it does, invalid connections are automatically removed when the connection pool is resized.

Connection Wrapping

Wrapping Connections

If the *Wrap JDBC Objects* option is `true` (the default), wrapped JDBC objects are returned for `Statement`, `PreparedStatement`, `CallableStatement`, `ResultSet`, and `DatabaseMetaData`.

This option ensures that `Statement.getConnection()` is the same as `DataSource.getConnection()`. Therefore, this option should be `true` when both `Statement.getConnection()` and `DataSource.getConnection()` are done.

You can specify the *Wrap JDBC Objects* option in the following ways:

- Check or uncheck the *Wrap JDBC Objects* box on the *Edit Connection Pool Advanced Attributes* page in the Administration Console.
- Specify the `--wrapjdbcobjects` option in the `asadmin create-jdbc-connection-pool` command.

Obtaining a Physical Connection from a Wrapped Connection

The default `DataSource` implementation in Payara Server provides a `getConnection` method that retrieves the JDBC driver's `SQLConnection` from Payara Server's `Connection` wrapper.

The method signature is as follows:

```
import java.sql.Connection;

public class Connection{
    public Connection getConnection(Connection con) throws java.sql.SQLException;
}
```

For example:

```
import java.sql.Connection;

public class MyComponent{

    public void operation(){
        var ctx = new InitialContext();
        var ds = (com.sun.appserv.jdbc.DataSource)ctx.lookup("jdbc/MyBase");

        try(var con = ds.getConnection(); //return wrapped connection to pool on
close.
            var drivercon = ds.getConnection(con)){
            // Do db operations.
        }
    }
}
```

Using the `Connection.unwrap()` Method

Using the `Connection.unwrap()` method on a vendor-provided interface returns an object or a wrapper object implementing the vendor-provided interface, which the application can make use of to do vendor-specific database operations. Use the `Connection.isWrapperFor()` method on a vendor-provided interface to check whether the connection can provide an implementation of the vendor-provided interface.

Check your corresponding JDBC driver vendor's documentation for information on these interfaces.

Allowing Non-Component Callers

You can allow non-Jakarta-EE components, such as third party persistence managers, to use a managed JDBC connection pool.

The returned connection is automatically enlisted with the transaction context obtained from the transaction manager. Standard Jakarta EE components can also use such pools.

IMPORTANT	Connections obtained by non-component callers are not automatically closed at the end of a transaction by the container. They must be explicitly closed by the caller.

You can enable non-component callers in the following ways:

- Check the *Allow Non Component* Callers box on the *Edit Connection Pool Advanced Attributes* page in the Administration Console. The default is `false`.
- Specify the `--allownoncomponentcallers` option in the `asadmin create-jdbc-connection-pool` command.
- Specify the `allow-non-component-callers` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.allow-non-component-callers=true
```

- Create a JDBC resource with a `__pm` suffix.

Accessing a `DataSource` using the `Synchronization.beforeCompletion()` method requires setting *Allow Non Component Callers* to `true`. For more information about the Transaction Synchronization Registry, see the-transaction-manager-the-transaction-synchronization-registry-and-usertransaction.

Using Application-Scoped Resources

You can define an application-scoped database or other resources for an enterprise application, web module, EJB module, connector module, or application client module by supplying a `payara-resources.xml` deployment descriptor file.

For more details, see "application-scoped-resources" in the Payara Server Application Deployment section.

JDBC Development Guidelines

In Jakarta EE applications and above, a JDBC data source can be deployed by adding the `@DataSourceDefinition` annotation to a managed component (like and

`@Stateless` EJB bean for example). The **Log JDBC Calls** setting can be configured using this annotation as well:

```
@DataSourceDefinition(
    name = "java:app/MyApp/MyDS",
    className = "org.h2.jdbcx.JdbcDataSource",
    url = "jdbc:h2:mem:test",
    properties = {"fish.payara.log-jdbc-calls=true"})
public class JDBCComponent{

}
```

The Data source definition can also be added to a deployment descriptor of an application, for example in the `web.xml` standard deployment descriptor:

```
<data-source>
   <name>java:global/ExampleDataSource</name>
   <class-name>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</class-name>
   <server-name>localhost</server-name>
   <port-number>3306</port-number>
   <database-name>mysql</database-name>
   <user>test</user>
   <password>test</password>
   <!-- Example of how to use a Payara Server specific custom connection pool
setting -->
   <property>
       <name>fish.payara.log-jdbc-calls</name>
       <value>true</value>
   </property>
</data-source>
```

Or in the definition of a `jdbc-connection-pool` in a `payara-resources.xml` file:

```
<jdbc-connection-pool name="examplePool"
        res-type="javax.sql.DataSource"
        datasource-classname="org.h2.jdbcx.JdbcDataSource" log-jdbc-calls="true">
    <property name="user" value="test"/>
    <property name="password" value="test"/>
    <property name="url" value="java:global/ExampleDataSource"/>
</jdbc-connection-pool>
```

This can also be done in the creation of a JDBC Connection Pool with the `--logjdbccalls` option.

```
asadmin create jdbc-connection-pool --datasourceclassname
org.h2.jdbcx.JdbcDataSource --restype javax.sql.XADataSource --logjdbccalls=true
examplePool
```

Likewise, you can configure the Slow Query Log Threshold time by setting the property `fish.payara.slow-query-threshold-in-seconds`, SQL Trace Listener classes by setting the property `fish.payara.sql-trace-listeners` in the `@DataSourceDefinition` annotation.

The below example shows the Slow Query Log Threshold and SQL Trace Listener set in the web.xml file.

```
<data-source>
   <name>java:global/ExampleDataSource</name>
   <class-name>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</class-name>
   <server-name>localhost</server-name>
   <port-number>3306</port-number>
   <database-name>mysql</database-name>
   <user>test</user>
   <password>test</password>
   <!-- Example of how to use a Payara specific custom connection pool setting -->
   <property>
       <name>fish.payara.log-jdbc-calls</name>
       <value>true</value>
   </property>
   <property>
       <name>fish.payara.slow-query-threshold-in-seconds</name>
       <value>5</value>
   </property>
   <property>
       <name>fish.payara.sql-trace-listeners</name>
       <value>
fish.payara.examples.payaramicro.datasource.example.CustomSQLTracer</value>
   </property>
</data-source>
```

Slow Query Trace Example

Below is an example `WARNING` trace for a slow query recollected from the server's log:

```
[#|2016-02-01T22:39:29.289+0000|WARNING|Payara
4.1|jakarta.enterprise.resource.sqltrace.com.sun.gjc.util|_ThreadID=61;_ThreadName=
http-listener-1(2);_TimeMillis=1454366369289;_LevelValue=900;|
  SQL Query Exceeded Threshold Time: 5000(ms): Time Taken: 10000(ms)
Query was SELECT ID, AGE, BIO, BIRTHDATE, BIRTHDAY, DATEFORMAT, DATEOFBIRTH,
DATEOFHIRE, EMAIL, HIREDATE, HIREDAY, MEMBERAGE, NAME, TODAYSDATE FROM MEMBERENTITY
WHERE (NAME = ?);

java.lang.Exception: Stack Trace shows code path to SQL
    at fish.payara.jdbc.SlowSQLLogger.sqlTrace(SlowSQLLogger.java:123)
    at com.sun.gjc.util.SQLTraceDelegator.sqlTrace(SQLTraceDelegator.java:122)
    at
com.sun.gjc.spi.jdbc40.ProfiledConnectionWrapper40$1.invoke(ProfiledConnectionWrapp
er40.java:448)
    at com.sun.proxy.$Proxy265.executeQuery(Unknown Source)
```

```
    at
org.eclipse.persistence.internal.databaseaccess.DatabaseAccessor.executeSelect(Data
baseAccessor.java:1009)
    at
org.eclipse.persistence.internal.databaseaccess.DatabaseAccessor.basicExecuteCall(D
atabaseAccessor.java:644)
    at
org.eclipse.persistence.internal.databaseaccess.DatabaseAccessor.executeCall(Databa
seAccessor.java:560)
    at
org.eclipse.persistence.internal.sessions.AbstractSession.basicExecuteCall(Abstract
Session.java:2055)
    at
org.eclipse.persistence.sessions.server.ServerSession.executeCall(ServerSession.jav
a:570)
    at
org.eclipse.persistence.internal.queries.DatasourceCallQueryMechanism.executeCall(D
atasourceCallQueryMechanism.java:242)
    at
org.eclipse.persistence.internal.queries.DatasourceCallQueryMechanism.executeCall(D
atasourceCallQueryMechanism.java:228)
    at
org.eclipse.persistence.internal.queries.DatasourceCallQueryMechanism.executeSelect
Call(DatasourceCallQueryMechanism.java:299)
    at
org.eclipse.persistence.internal.queries.DatasourceCallQueryMechanism.selectAllRows
(DatasourceCallQueryMechanism.java:694)
    at
org.eclipse.persistence.internal.queries.ExpressionQueryMechanism.selectAllRowsFrom
Table(ExpressionQueryMechanism.java:2740)
    at
org.eclipse.persistence.internal.queries.ExpressionQueryMechanism.selectAllRows(Exp
ressionQueryMechanism.java:2693)
    at
org.eclipse.persistence.queries.ReadAllQuery.executeObjectLevelReadQuery(ReadAllQue
ry.java:559)
    at
org.eclipse.persistence.queries.ObjectLevelReadQuery.executeDatabaseQuery(ObjectLev
elReadQuery.java:1175)
    at
org.eclipse.persistence.queries.DatabaseQuery.execute(DatabaseQuery.java:904)
    at
org.eclipse.persistence.queries.ObjectLevelReadQuery.execute(ObjectLevelReadQuery.j
ava:1134)
    at org.eclipse.persistence.queries.ReadAllQuery.execute(ReadAllQuery.java:460)
    at
org.eclipse.persistence.queries.ObjectLevelReadQuery.executeInUnitOfWork(ObjectLeve
lReadQuery.java:1222)
    at
org.eclipse.persistence.internal.sessions.UnitOfWorkImpl.internalExecuteQuery(UnitO
fWorkImpl.java:2896)
    at
org.eclipse.persistence.internal.sessions.AbstractSession.executeQuery(AbstractSess
ion.java:1857)
    at
org.eclipse.persistence.internal.sessions.AbstractSession.executeQuery(AbstractSess
ion.java:1839)
    at
org.eclipse.persistence.internal.sessions.AbstractSession.executeQuery(AbstractSess
ion.java:1804)
```

```
    at
org.eclipse.persistence.internal.jpa.QueryImpl.executeReadQuery(QueryImpl.java:258)
    at
org.eclipse.persistence.internal.jpa.QueryImpl.getResultList(QueryImpl.java:473)
    at
fish.payara.team.info.controllers.MemberSessionBean.getTeamMemberByName(MemberSessi
onBean.java:35)
```

Using the Transaction Service

The Jakarta EE platform provides several abstractions that simplify development of dependable transaction processing for applications. This chapter discusses Jakarta EE transactions and transaction support in Azul Payara.

For more information about the Jakarta Transactions API (JTA) and Java Transaction Service (JTS), see "administering-transactions" in Payara Server General Administration section and in the Jakarta Transactions specification.

You might also want to read "Transactions" in The Jakarta EE Tutorial.

Handling Transactions with Databases

Using JDBC Transaction Isolation Levels

Not all database vendors support all transaction isolation levels available in the JDBC API. Payara Server allows specifying any isolation level your database supports. The following table defines transaction isolation levels.

Table 196. Transaction Isolation Levels

Transaction Isolation Level	`getTransactionIsolation` Return Value	Description
`read-uncommitted`	`1`	Dirty reads, non-repeatable reads, and phantom reads can occur.
`read-committed`	`2`	Dirty reads are prevented; non-repeatable reads and phantom reads can occur.
`repeatable-read`	`4`	Dirty reads and non-repeatable reads are prevented; phantom reads can occur.
`serializable`	`8`	Dirty reads, non-repeatable reads and phantom reads are prevented.

By default, the transaction isolation level is undefined (empty), and the JDBC driver's default isolation level is used. You can specify the transaction isolation level in the following ways:

- Select the value from the *Transaction Isolation* drop-down list on the *New JDBC Connection Pool* or *Edit Connection Pool* page in the Administration Console.
- Specify the `--isolationlevel` option in the `asadmin create-jdbc-connection-pool` command.
- Specify the `transaction-isolation-level` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.H2Pool.transaction-isolation-level=serializable
```

NOTE You cannot execute the `setTransactionIsolation` method during a transaction.

You can set the default transaction isolation level for a JDBC connection pool. For details, see "to-create-a-jdbc-connection-pool" in the Payara Server General Administration section.

To verify that a level is supported by your database management system, test your database programmatically using the `supportsTransactionIsolationLevel` method in `java.sql.DatabaseMetaData`, as shown in the following example:

```
var ctx = new InitialContext();
var ds = (DataSource) ctx.lookup("jdbc/MyDatasource");
var con = ds.getConnection();
var dbmd = con.getMetaData();

if (dbmd.supportsTransactionIsolationLevel(TRANSACTION_SERIALIZABLE){
    Connection.setTransactionIsolation(TRANSACTION_SERIALIZABLE);
}
```

For more information about these isolation levels and what they mean, see the JDBC API specification.

WARNING	Setting or resetting the transaction isolation level for every `getConnection` call can degrade performance. So by default, the isolation level is not guaranteed.

Applications that change the transaction isolation level on a pooled connection programmatically risk polluting the JDBC connection pool, which can lead to errors. If an application changes the isolation level, enabling the `is-isolation-level-guaranteed` setting in the pool can minimize such errors.

You can guarantee the transaction isolation level in the following ways:

- Check the *Isolation Level Guaranteed* box on the *New JDBC Connection Pool* or *Edit Connection Pool* page in the Administration Console.
- Specify the `--isisolationguaranteed` option in the `asadmin create-jdbc-connection-pool` command.
- Specify the `is-isolation-level-guaranteed` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.H2Pool.is-isolation-level-guaranteed=true
```

Using Non-Transactional Connections

You can specify a non-transactional database connection in any of these ways:

- Check the *Non-Transactional Connections* box on the N*ew JDBC Connection Pool* or *Edit Connection Pool* page in the Administration Console.
- Specify the `--nontransactionalconnections` option in the `asadmin create-jdbc-connection-pool` command.
- Specify the `non-transactional-connections` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.H2Pool.non-transactional-connections=true
```

- Use Azul Payara's `DataSource` implementation, which provides a `getNonTxConnection` method. This method retrieves a JDBC connection that is not in the scope of any transaction. There are two variants:

```
public java.sql.Connection getNonTxConnection() throws java.sql.SQLException

public java.sql.Connection getNonTxConnection(String user, String password)
throws java.sql.SQLException
```

- Create a resource with the JNDI name ending in the `__nontx` suffix. This forces all connections looked up using this resource to be non-transactional.

Typically, a connection is enlisted in the context of the transaction in which a `getConnection` call is invoked. However, a non-transactional connection is not enlisted in a transaction context even if a transaction is in progress.

The main advantage of using non-transactional connections is that the overhead incurred in enlisting and de-listing connections in transaction contexts is avoided.

CAUTION	Use these connections carefully.

For example, if a non-transactional connection is used to query the database while a transaction is in progress that modifies the database, the query retrieves the unmodified data in the database.

This is because the in-progress transaction hasn't committed. For another example, if a non-transactional connection modifies the database and a transaction that is running simultaneously rolls back, the changes made by the non-transactional connection are not rolled back.

Here is a typical use case for a non-transactional connection: a component that is updating a database in a transaction context spanning over several iterations of a loop can refresh cached data by using a non-transactional connection to read data before the transaction commits.

Handling Transactions with Enterprise Beans

This section describes the transaction support built into the Jakarta Enterprise Beans programming model.

As a developer, you can write an application that updates data in multiple databases distributed across multiple sites. The application might use EJB servers from different vendors.

Flat Transactions

The Jakarta Enterprise Beans specification support for flat (as opposed to nested) transactions. In a flat transaction, each transaction is decoupled and remains independent of other transactions in the system. Another transaction cannot start in the same thread until the current transaction ends.

Flat transactions are the most prevalent model and are supported by most commercial database systems. Although nested transactions offer a finer granularity of control over transactions, they are supported by far fewer commercial database systems.

Global and Local Transactions

Both local and global transactions are demarcated using the `jakarta.transaction.UserTransaction` interface, which the client must use. Local transactions bypass the XA commit protocol and are faster.

For more information, see The Transaction Manager, the Transaction Synchronization Registry, and `UserTransaction`.

Commit Options

The EJB specification's transaction guidelines are designed to give the container the flexibility to select the disposition of the instance state at the time a transaction is committed. This allows the container to best manage caching an entity object's state and associating an entity object identity with the EJB instances.

There are three commit-time options:

Option A: The container caches a ready instance between transactions. The container

ensures that the instance has exclusive access to the state of the object in persistent storage.

+ In this case, the container does not have to synchronize the instance's state from the persistent storage at the beginning of the next transaction.

+ IMPORTANT: Commit option A is not supported in Azul Payara's EJB implementation.

Option B: The container caches a ready instance between transactions, but the container does not ensure that the instance has exclusive access to the state of the object in persistent storage.

+ NOTE: This is the default commit option set in Azul Payara.

+ In this case, the container must synchronize the instance's state by invoking the `ejbLoad` lifecycle method from persistent storage at the beginning of the next transaction.

Option C: The container does not cache a ready instance between transactions, but instead returns the instance to the pool of available instances after a transaction has completed.

+ The life cycle for every business method invocation under commit option C looks like this.

+

```
ejbActivate   ejbLoad   business method   ejbStore   ejbPassivate
```

+ If there is more than one transactional client concurrently accessing the same entity, the first client gets the ready instance and subsequent concurrent clients get new instances from the pool.

The `payara-ejb-jar.xml` deployment descriptor has an element, `commit-option`, that specifies the commit option to be used. Based on the specified commit option, the appropriate handler is instantiated.

Bean-Level Container-Managed Transaction Timeouts

The transaction timeout for the domain is specified using the *Transaction Timeout* setting of the server's Transaction Service.

A transaction started by the container must commit (or rollback) within this time, regardless of whether the transaction is suspended (and resumed), or the transaction is marked for rollback. The default value, `0`, specifies that the server waits indefinitely for a transaction to complete.

To override this timeout for an individual bean, use the optional `cmt-timeout-in-seconds` element in `payara-ejb-jar.xml`. The default value, `0`, specifies that the Transaction Service timeout is used.

The value of `cmt-timeout-in-seconds` is used for all methods in the bean that start a new container-managed transaction.

IMPORTANT	This timeout value is not used if the bean joins a client transaction.

Handling Transactions with Jakarta Messaging

Transactions and Non-Persistent Messages

During transaction recovery, non-persistent messages might be lost. If the broker fails between the transaction manager `prepare` and `commit` operations, any non-persistent message in the transaction is lost and cannot be delivered.

A message that is not saved to a persistent store is not available for transaction recovery.

Using the Configurabletransactionsupport Interface

The Jakarta Connectors specification allows a resource adapter to use the `transaction-support` attribute to specify the level of transaction support that the resource adapter handles.

However, the resource adapter vendor does not have a mechanism to figure out the current transactional context in which a `ManagedConnectionFactory` is used.

If a `ManagedConnectionFactory` implements an optional interface called `com.sun.appserv.connectors.spi.ConfigurableTransactionSupport`, Payara Server notifies the `ManagedConnectionFactory` of the `transaction-support` configured for the connector connection pool when the `ManagedConnectionFactory` instance is created for the pool.

Connections obtained from the pool can then be used with a transaction level at or lower than the configured value. For example, a connection obtained from a pool that is set to `XA_TRANSACTION` could be used as a `LOCAL` resource in a last-agent-optimized transaction or in a non-transactional context.

The Transaction Manager, the Transaction Synchronization Registry, and `UserTransaction`

To access a `UserTransaction` instance, you can either look it up using the `java:comp/UserTransaction` JNDI name or inject it using the `@Resource` annotation.

Accessing a `DataSource` using the `Synchronization.beforeCompletion()` method requires setting the *Allow Non-Component Callers* option of its corresponding JDBC Connection Pool to `true`.

The default setting is `false`. For more information about non-component callers, see allowing-non-component-callers.

If possible, you should use the `jakarta.transaction.TransactionSynchronizationRegistry` interface instead of `jakarta.transaction.TransactionManager` , for portability. You can look up the implementation of this interface by using the JNDI name `java:comp/TransactionSynchronizationRegistry`.

For details, see the `TransactionSynchronizationRegistryInterface` API documentation and the Jakarta Transactions specification.

If accessing the `jakarta.transaction.TransactionManager` implementation is

absolutely necessary, you can look up Azul Payara's implementation of this interface using the JNDI name `java:appserver/TransactionManager`.

CAUTION	This lookup should not be used by application code under any circumstances.

Using the Java Naming and Directory Interface

A naming service maintains a set of bindings, which relate names to objects. Azul Payara's naming service is based on the Java Naming and Directory Interface (JNDI) API. The JNDI API allows application components and clients to look up distributed resources, services, and EJB components.

For general information about the JNDI API, see `http://www.oracle.com/technetwork/java/jndi/index.html`.

IMPORTANT	Payara Server's Web Profile supports the EJB 4.0 Lite specification, which allows enterprise beans within web applications, among other features. The Full Profile supports the entire EJB 4.0 specification. For details, see Jakarta Enterprise Beans.

Accessing the Naming Context

Payara Server provides a naming environment, or context, which is compliant with standard Jakarta EE requirements. A `Context` object provides the methods for binding names to objects, unbinding names from objects, renaming objects, and listing the bindings.

The `InitialContext` is the handle to the Jakarta EE naming service that application components and clients use for lookups.

The JNDI API also provides subcontext functionality. Much like a directory in a file system, a subcontext is a context within a context. This hierarchical structure permits better organization of information. For naming services that support subcontexts, the

`Context` class also provides methods for creating and destroying subcontexts.

NOTE	Each resource within a server instance must have a unique name. However, two resources in different server instances or different domains can have the same name.

Portable Global JNDI Names

If an EJB component is a kind of session bean, and it is deployed to any implementation supporting the Jakarta Enterprise Beans 4.0 specification, it automatically has one or more portable JNDI names defined based on the syntax in the specification.

NOTE	This is true of existing EJB 3.0 and 2.x applications that are deployed to an implementation supporting EJB 3.1 or newer versions.

No code changes are required to the bean class itself in order to have the portable global JNDI name automatically assigned when deployed to an EJB container.

For more information, see the Jakarta EE Platform Specification, section 5.2.2, "Application Component Environment Namespaces", and the EJB 4.0 Specification, section 4.4. Access in the Global JNDI Namespace.

NOTE	If the `disable-nonportable-jndi-names` property is set to `false` (the default), a Payara Server JNDI name is assigned in addition to a portable global JNDI name.

Payara Server Vendor-Specific Global JNDI Names

Payara Server vendor-specific global JNDI names are assigned according to the following precedence rules:

1. A global JNDI name assigned in the `payara-ejb-jar.xml`, `payara-web.xml`, or `payara-application-client.xml` deployment descriptor file has the highest precedence. See Mapping References.
2. A global JNDI name assigned in a `mapped-name` element in the `ejb-jar.xml`,

`web.xml`, or `application-client.xml` deployment descriptor file has the second-highest precedence. The following elements have `mapped-name` sub-elements: `resource-ref`, `resource-env-ref`, `ejb-ref`, `message-destination`, `message-destination-ref`, `session`, `message-driven`, and `entity`.

3. A global JNDI name assigned in a `mappedName` attribute of an annotation has the third-highest precedence. The following annotations have `mappedName` attributes: `@jakarta.annotation.Resource`, `@jakarta.ejb.EJB`, `@jakarta.ejb.Stateless`, `@jakarta.ejb.Singleton`, `@jakarta.ejb.Stateful`, and `@javax.ejb.MessageDriven`.
4. In most cases, a default global JNDI name is assigned (and recorded in the server log) if no name is assigned in deployment descriptors or annotations.
    - For a session or entity bean, a Payara Server specific JNDI name is assigned as follows:
        - For an EJB 2.x dependency or a session or entity bean with a remote interface, the default is the fully qualified name of the home interface.
        - For an EJB 3.0+ dependency or a session bean with a remote interface, the default is the fully qualified name of the remote business interface.
        - If both EJB 2.x and EJB 3.0 remote interfaces are specified, or if more than one 3.0 remote interface is specified, there is no Payara Server specific default.

          For an entity bean, a global JNDI name must be assigned.
    - For all other component dependencies that must be mapped to global JNDI names, the default is the name of the dependency relative to `java:comp/env`.

      For example, in the `@Resource(name="jdbc/Foo") DataSource ds;` annotation, the global JNDI name is `jdbc/Foo`.

Disabling Payara Server JNDI Names

The EJB specification supported by Payara Server defines portable EJB JNDI names for

session beans. Because of this, there is less need to continue to use older vendor-specific JNDI names.

By default, Payara Server specific JNDI names are generated automatically for backward compatibility. However, this can lead to some ease-of-use issues.

For example, deploying two different applications containing a remote EJB component that exposes the same remote interface causes a conflict between the default JNDI names.

The default handling of JNDI specific names in Payara Server can be managed by using the `asadmin` command:

```
asadmin> set server.ejb-container.property.disable-nonportable-jndi-names="true"
```

The `disable-nonportable-jndi-names` property is a boolean flag that can take the following values:

- **`false`**: Enables the automatic use of Payara Server specific JNDI names in addition to portable global JNDI names. This is the default setting.
- **`true`**: Disables the automatic use of specific JNDI names. In all cases, only portable global JNDI names are used.

TIP	This setting applies to all session beans deployed in the server.

Accessing EJB Components Using the `CosNaming` Naming Context

The preferred way of accessing the naming service, even in code that runs outside a Jakarta EE container, is to use the no-argument `InitialContext` constructor.

However, if EJB client code explicitly instantiates an `InitialContext` that points to the `CosNaming` naming service, it is necessary to set the `java.naming.factory.initial` property to `org.glassfish.jndi.cosnaming.CNCtxFactory` in the client JVM software when accessing EJB components.

You can set this property using the `asadmin create-jvm-options` command, as follows:

```
asadmin> create-jvm-options -Djava.naming.factory.initial
=org.glassfish.jndi.cosnaming.CNCtxFactory
```

Or you can set this property in the code, as follows:

```
try {
    var properties = new Properties();
    properties.put("java.naming.factory.initial",
"org.glassfish.jndi.cosnaming.CNCtxFactory");
}
```

The `java.naming.factory.initial` property applies to only one instance. The property is neither cluster or deployment group-aware.

Accessing EJB Components in a Remote Payara Server

The recommended approach for looking up an EJB component in a remote Payara Server from a client that is a servlet or EJB component is to use the Interoperable Naming Service syntax.

Host and port information is prepended to any global JNDI names and is automatically resolved during the lookup. The syntax for an interoperable global name is as follows:

```
corbaname:iiop:${host}:${port}#{ejbName}
```

This makes the programming model for accessing EJB components in another Payara Server exactly the same as accessing them in the same server. The deployer can change the way the EJB components are physically distributed without having to change the code.

For Jakarta EE components, the code still performs a `java:comp/env` lookup on an EJB reference. The only difference is that the deployer maps the `ejb-ref` element to an interoperable name in a Payara Server deployment descriptor file instead of to a simple global JNDI name.

For example, suppose a servlet looks up an EJB reference using `java:comp/env/ejb/Foo`, and the target EJB component has a global JNDI name of `a/b/Foo`.

The `ejb-ref` element in `glassfish-web.xml` or `payara-web.xml` looks like this:

```
<ejb-ref>
   <ejb-ref-name>ejb/Foo</ejb-ref-name>
   <jndi-name>corbaname:iiop:host:port#a/b/Foo</jndi-name>
</ejb-ref>
```

The code looks like this:

```
Context ic = new InitialContext();
Object o = ic.lookup("java:comp/env/ejb/Foo");
```

For a client that doesn't run within a Jakarta EE container, the code just uses the interoperable global name instead of the simple global JNDI name. For example:

```
Context ic = new InitialContext();
Object o = ic.lookup("corbaname:iiop:host:port#a/b/Foo");
```

IMPORTANT	Objects stored in the interoperable naming context and component-specific (`java:comp/env`) naming contexts are transient. On each server startup or application reloading, all relevant objects are re-bound to the namespace.
IMPORTANT	glassfish-web.xml is deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

Configuring Resources

Payara Server exposes special resources in the naming context environment.

External JNDI Resources

An external JNDI resource defines custom JNDI contexts and implements the `javax.naming.spi.InitialContextFactory` interface. There is no specific JNDI

parent context for external JNDI resources, except for the standard `java:comp/env/`.

Create an external JNDI resource in one of these ways:

- To create an external JNDI resource using the Administration Console, open the *Resources* component, open the *JNDI* component, and select *External Resources*.
- To create an external JNDI resource, use the `asadmin create-jndi-resource` command.

Custom Resources

A custom resource specifies a custom server-wide resource object factory that implements the `javax.naming.spi.ObjectFactory` interface. There is no specific JNDI parent context for external JNDI resources, except for the standard `java:comp/env/`.

Create a custom resource in one of these ways:

- To create a custom resource using the Administration Console, open the *Resources* component, open the *JNDI* component, and select *Custom Resources*.
- To create a custom resource, use the `asadmin create-custom-resource` command.

Built-In Factories for Custom Resources

Payara Server provides built-in factories for the following types of custom resources:

- JavaBeans
- Properties
- Java Primitives and String objects
- URLs

It is possible to create custom resources by using a `payara-resources.xml` or `glassfish-resources.xml` deployment descriptor. To this effect, several template files for these built-in factories and a `README` file are available at `as-`

`install/lib/install/templates/resources/custom/`.

IMPORTANT	glassfish-resources.xml is deprecated and support will be removed in a future release. It is recommended to use payara-resources.xml instead.

Javabeanfactory

To create a custom resource that provides instances of a JavaBean class, follow these steps:

1. Set the custom resource's factory class to `org.glassfish.resources.custom.factory.JavaBeanFactory`.
2. Create a property in the custom resource for each setter method in the JavaBean class.

   For example, if the JavaBean class has a method named `setAccount`, specify a property named `account` and give it a value.

3. Make sure the JavaBean class is accessible to the Payara Server.

   For example, you can place the JavaBean class in the `as-install/lib` directory.

Propertiesfactory

To create a custom resource that provides configuration properties to applications, set the custom resource's factory class to `org.glassfish.resources.custom.factory.PropertiesFactory`, then specify one or both of the following:

- Create a property in the custom resource named `org.glassfish.resources.custom.factory.PropertiesFactory.fileName` and specify as its value the path to a properties file or an XML file.

  The path can be absolute or relative to the installation directory. The file must be accessible to the operating system user running the Payara Server's process.

If an XML file is specified, it must match the document type definition (DTD) specified in the API definition of java.util.Properties.

- Create the desired properties directly as properties of the custom resource.

  If both the `fileName` property and other properties are specified, the resulting property set is the union of the contents of both.

  If the same property is defined in the file and directly in the custom resource, the value of the latter takes precedence.

Primitivesandstringfactory

To create a custom resource that provides Java primitives to applications, follow these steps:

1. Set the custom resource's factory class to `org.glassfish.resources.custom.factory.PrimitivesAndStringFactory`.
2. Set the custom resource's resource type to one of the following or its fully qualified wrapper class name equivalent:
   - `int`
   - `long`
   - `double`
   - `float`
   - `char`
   - `short`
   - `byte`
   - `boolean`
   - `String`
3. Create a property in the custom resource named `value` and give it the value needed

by the application.

For example, If the application requires a `double` of value `22.1`, create a property with the name `value` and the value `22.1`.

Urlfactory

To create a custom resource that provides URL instances to applications, follow these steps:

1. Set the custom resource's factory class to `org.glassfish.resources.custom.factory.URLObjectFactory`.
2. Choose which of the following constructors to use:
   - `URL(protocol, host, port, file)`
   - `URL(protocol, host, file)`
   - `URL(spec)`
3. Define properties according to the chosen constructor.

   For example, for the first constructor, define properties named `protocol`, `host`, `port`, and `file`. Example values might be `http`, `localhost`, `8085`, and `index.html`, respectively.

   For the third constructor, define a property named `spec` and assign it the value of the entire URL.

Using Application-Scoped Resources

You can define an application-scoped JNDI resource for an enterprise application, web module, EJB module, connector module, or application client module by supplying a `glassfish-resources.xml` or `payara-resources.xml` deployment descriptor file.

IMPORTANT	glassfish-resources.xml is deprecated and support will be removed in a future release. It is recommended to use payara-

> resources.xml instead.

For details, see "application-scoped-resources" in the Application Deployment section.

Using a Custom `jndi.properties` File

To use a custom `jndi.properties` file, JAR it and place it in the `domain-dir/lib` directory. This adds the custom `jndi.properties` file to the Common class loader. For more information about class loading, see "Class Loaders".

> **TIP** For each property found in more than one `jndi.properties` file, the Jakarta EE naming service either uses the first value found or concatenates all values, whichever makes sense at runtime.

Mapping References

The following XML elements in the Payara Server deployment descriptors map resource references in application client, EJB, and web application components to JNDI names configured in Payara Server:

`resource-env-ref`: Maps the `@Resource` or `@Resources` annotation (or the `resource-env-ref` element in the corresponding Jakarta EE XML file) to the absolute JNDI name configured in Payara Server. **`resource-ref`**: Maps the `@Resource` or `@Resources` annotation (or the `resource-ref` element in the corresponding Jakarta EE XML file) to the absolute JNDI name configured in Payara Server. **`ejb-ref`**: Maps the `@EJB` annotation (or the `ejb-ref` element in the corresponding Jakarta EE XML file) to the absolute JNDI name configured in Payara Server.

+ JNDI names for EJB components must be unique. For example, appending the application name and the module name to the EJB name is one way to guarantee unique names.

+ In this case, `mycompany.pkging.pkgingEJB.MyEJB` would be the JNDI name for an EJB in the module `pkgingEJB.jar`, which is packaged in the `pkging.ear` application.

These elements are part of the `glassfish-web.xml`/`payara-web.xml`, `glassfish-application-client.xml`/`payara-application-client.xml`, `glassfish-ejb-jar.xml`/`payara-ejb-jar.xml`, and `glassfish-application.xml`/`payara-application.xml` deployment descriptor files.

IMPORTANT	glassfish-* deployment descriptors are deprecated and support will be removed in a future release. It is recommended to use the payara-* equivalents instead.

For more information about how these elements behave in each of the deployment descriptor files, see "elements-of-the-payara-platform-deployment-descriptors" in the Payara Server Application Deployment section.

The following is an example of a JDBC resource lookup to showcase how to reference resource factories. The same principle is applicable to all resources (such as JMS destinations, JavaMail sessions, and so on).

The `@Resource` annotation in the application code looks like this:

```
@Resource(name="jdbc/helloDbDs")
javax.sql.DataSource ds;
```

This references a resource with the JNDI name of `java:jdbc/helloDbDs`. If this is the JNDI name of the JDBC resource configured in Payara Server, the annotation alone is enough to reference the resource.

However, you can use a Payara Server specific deployment descriptor to override the annotation. For example, the `resource-ref` element in the `payara-web.xml` file maps the `res-ref-name` (the name specified in the annotation) to the JNDI name of another JDBC resource configured in Payara Server.

```
<resource-ref>
   <res-ref-name>jdbc/helloDbDs</res-ref-name>
   <jndi-name>jdbc/helloDbDataSource</jndi-name>
</resource-ref>
```

Using Jakarta Messaging

This chapter describes how to use the Jakarta Messaging (JMS) API. Payara Server has a fully integrated JMS provider: the **Open Message Queue** software.

IMPORTANT	JMS resources are supported only in the Full Profile, not in the Web Profile.

For information about the JMS, see "Messaging" in The Jakarta EE Tutorial.

For detailed information about JMS concepts and JMS support in Payara Server, see "administering-the-java-message-service-jms" in the Payara Server General Administration section.

Using Application-Scoped JMS Resources

You can define an application-scoped JMS or other resource for an enterprise application, web module, EJB module, connector module, or application client module by supplying a `payara-resources.xml` deployment descriptor file. For details, see "application-scoped-resources" in the Payara Server Application Deployment section.

Load-Balanced Message Inflow

You can configure `ActivationSpec` properties of the `jmsra` resource adapter in the `payara-ejb-jar.xml` file for a message-driven bean using `activation-config-property` elements. Whenever a message-driven bean (`EndPointFactory`) is deployed, the connector runtime engine finds these properties and configures them accordingly in the resource adapter.

See "activation-config-property" in the Payara Server Application Deployment section.

Payara Server transparently enables messages to be delivered in random fashion to message-driven beans having the same `ClientID`.

NOTE	The `ClientID` is required for durable subscribers. For non-durable subscribers in which the `ClientID` is not configured, all instances of a specific message-driven bean that subscribe to same

> topic are considered equal.

When a message-driven bean is deployed to multiple instances of Payara Server, only one of the message-driven beans receives the message. If multiple distinct message-driven beans subscribe to same topic, one instance of each message-driven bean receives a copy of the message.

To support multiple consumers using the same queue, set the `maxNumActiveConsumers` property of the physical destination to a large value. If this property is set, the **Open Message Queue** software allows multiple message-driven beans to consume messages from the same queue.

The message is delivered randomly to the message-driven beans. If `maxNumActiveConsumers` is set to `-1`, there is no limit to the number of consumers.

To ensure that local delivery is preferred, set `addresslist-behavior` to `priority`. This setting specifies that the first broker in the `AddressList` is selected first.

This first broker is the local co-located message queue instance. If this broker is unavailable, connection attempts are made to brokers in the order in which they are listed in the `AddressList`. This setting is the default for Payara Server instances that belong to a cluster or deployment group.

Authentication with `ConnectionFactory`

If your web, EJB, or client module has `res-auth` set to `Container`, but you use the `ConnectionFactory.createConnection("user","password")` method to get a connection, Payara Server searches the container for authentication information before using the supplied user and password.

Delivering SOAP Messages Using the JMS API

Web service clients use the Simple Object Access Protocol (SOAP) to communicate with web services. SOAP uses a combination of XML-based data structuring and Hyper Text Transfer Protocol (HTTP) to define a standardized way of invoking methods in objects distributed in diverse operating environments across the Internet.

You can take advantage of the JMS provider's reliable messaging when delivering SOAP messages. You can convert a SOAP message into a JMS message, send the JMS message, then convert the JMS message back into a SOAP message.

To Send SOAP Messages Using the JMS API

1. Import the `MessageTransformer` library.

   ```
   import com.sun.messaging.xml.MessageTransformer;
   ```

   This is the utility class whose methods you use to convert SOAP messages to JMS messages and vice-versa. You can then send a JMS message containing a SOAP payload as if it were a normal JMS message.

2. Initialize the `TopicConnectionFactory`, `TopicConnection`, `TopicSession`, and publisher components.

   ```
   var tcf = new TopicConnectionFactory();
   var tc = tcf.createTopicConnection();
   var session = tc.createTopicSession(false,Session.AUTO_ACKNOWLEDGE);
   var topic = session.createTopic(topicName);
   var publisher = session.createPublisher(topic);
   ```

3. Construct a SOAP message using the **SOAP with Attachments API** for Java (SAAJ).

   ```
   /*construct a default soap MessageFactory */
   MessageFactory mf = MessageFactory.newInstance();
   * Create a SOAP message object.*/
   SOAPMessage soapMessage = mf.createMessage();
   /** Get SOAP part.*/
   SOAPPart soapPart = soapMessage.getSOAPPart();
   /* Get SOAP envelope. */
   SOAPEnvelope soapEnvelope = soapPart.getEnvelope();
   /* Get SOAP body.*/
   SOAPBody soapBody = soapEnvelope.getBody();
   /* Create a name object. with name space */
   /* http://www.sun.com/imq. */
   Name name = soapEnvelope.createName("HelloWorld", "hw",
    "http://www.sun.com/imq");
   * Add child element with the above name. */
   SOAPElement element = soapBody.addChildElement(name)
   /* Add another child element.*/
   element.addTextNode( "Welcome to Payara Web Services." );
   /* Create an atachment with activation API.*/
   URL url = new URL ("http://java.sun.com/webservices/");
   DataHandler dh = new DataHandler (url);
   AttachmentPart ap = soapMessage.createAttachmentPart(dh);
   ```

```
/*set content type/ID. */
ap.setContentType("text/html");
ap.setContentId("cid-001");
/** add the attachment to the SOAP message.*/
soapMessage.addAttachmentPart(ap);
soapMessage.saveChanges();
```

4. Convert the SOAP message to a JMS message by calling the `MessageTransformer.SOAPMessageintoJMSMessage()` method.

```
Message m = MessageTransformer.SOAPMessageIntoJMSMessage(soapMessage, session);
```

5. Publish the JMS message.

```
publisher.publish(m);
```

6. Close the JMS connection.

```
tc.close();
```

To Receive SOAP Messages Using the JMS API

1. Import the `MessageTransformer` library.

```
import com.sun.messaging.xml.MessageTransformer;
```

   This is the utility class whose methods you use to convert SOAP messages to JMS messages and vice-versa. The JMS message containing the SOAP payload is received as if it were a normal JMS message.

2. Initialize the `TopicConnectionFactory`, `TopicConnection`, `TopicSession`, `TopicSubscriber`, and Topic.

```
var messageFactory = MessageFactory.newInstance();
var tcf = new com.sun.messaging.TopicConnectionFactory();
var tc = tcf.createTopicConnection();
var session = tc.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
var topic = session.createTopic(topicName);
var subscriber = session.createSubscriber(topic);
var subscriber.setMessageListener(this);
tc.start();
```

3. Use the `OnMessage` method to receive the message. Use the `SOAPMessageFromJMSMessage` method to convert the JMS message to a SOAP message.

```
public void onMessage (Message message) {
    var soapMessage = MessageTransformer.SOAPMessageFromJMSMessage(message
,messageFactory);
}
```

4. Retrieve the content of the SOAP message.

Using Jakarta Mail

This chapter describes how to use the Jakarta Mail API, which provides a set of abstract classes defining objects that comprise a mail system.

IMPORTANT	Jakarta Mail resources are supported only in the Full Profile, not in the Web Profile.

Introducing Jakarta Mail

The Jakarta Mail API defines classes such as `Message`, `Store`, and `Transport`. The API can be extended and can be subclassed to provide new protocols and to add functionality when necessary. In addition, the API provides concrete subclasses of the abstract classes. These subclasses, including `MimeMessage` and `MimeBodyPart`, implement widely used Internet mail protocols and conform to the **RFC822** and **RFC2045** specifications. The Jakarta Mail API includes support for the **IMAP4**, **POP3**, and **SMTP** protocols.

The Jakarta Mail architectural components are as follows:

- The abstract layer declares classes, interfaces, and abstract methods intended to support mail handling functions that all mail systems support.
- The internet implementation layer implements part of the abstract layer using the **RFC822** and **MIME** internet standards.
- Jakarta Mail uses Jakarta Activation to encapsulate message data and to handle

commands intended to interact with that data.

For more information, see ""Administering the Jakarta Mail Service"" in the Payara Server General Administration section and the Jakarta Mail specification.

Creating a Jakarta Mail Session

You can create a Jakarta Mail session in the following ways:

- In the Administration Console, open the *Resources* component and select *JavaMail Sessions*.
- Use the `asadmin create-mail-resource` command.

Jakarta Mail Session Properties

You can set properties for a Jakarta Mail `Session` object. Every property name must start with a `mail-` prefix. Payara Server changes the dash (`-`) character to a period (`.`) in the name of the property and saves the property to the `MailConfiguration` and Jakarta Mail `Session` objects. If the name of the property doesn't start with `mail-`, the property is ignored.

For example, if you want to define the property `mail.from` in a Jakarta Mail `Session` object, first define the property as follows:

- Name: `mail-from`
- Value: `john.doe@sun.com`

Looking up a Jakarta Mail Session

The standard Java Naming and Directory Interface (JNDI) subcontext for Jakarta Mail sessions is `java:comp/env/mail`.

Registering Jakarta Mail sessions in the `mail` naming subcontext of a JNDI namespace, or in one of its child subcontexts, is a standard requirement.

The JNDI namespace is hierarchical, like a file system's directory structure, so it is easy to find and nest references. A Jakarta Mail session is bound to a logical JNDI name.

The name identifies a subcontext, `mail`, of the root context, and a logical name. To change the Jakarta Mail session, you can change its entry in the JNDI namespace without having to modify the application.

The resource lookup in the application code looks like this:

```
var ic = new InitialContext();
var snName = "java:comp/env/mail/MyMailSession";
var session = (Session)ic.lookup(snName);
```

For more information about the JNDI API, see using-a-custom-jndi.properties-file.

Sending and Reading Messages Using Jakarta Mail

To Send a Message Using Jakarta Mail

1. Import the packages that you need.

```
import java.util.*;
import jakarta.activation.*;
import jakarta.mail.*;
import jakarta.mail.internet.*;
import javax.naming.*;
```

2. Look up the Jakarta Mail session.

```
var ic = new InitialContext();
var snName = "java:comp/env/mail/MyMailSession";
var session = (Session)ic.lookup(snName);
```

3. Override the Jakarta Mail session properties if necessary. For example:

```
var props = session.getProperties();
props.put("mail.from", "user2@mailserver.com");
```

4. Create a `MimeMessage`. The `msgRecipient`, `msgSubject`, and `msgTxt` variables in the following example contain input from the user:

```
var msg = new MimeMessage(session);
msg.setSubject(msgSubject);
msg.setSentDate(new Date());
msg.setFrom();
msg.setRecipients(Message.RecipientType.TO, InternetAddress.parse(msgRecipient,
```

```
false));
msg.setText(msgTxt);
```

5. Send the message.

```
Transport.send(msg);
```

To Read a Message Using Jakarta Mail

1. Import the packages that you need.

```
import java.util.*;
import jakarta.activation.*;
import jakarta.mail.*;
import jakarta.mail.internet.*;
import javax.naming.*;
```

2. Look up the Jakarta Mail session.

```
var ic = new InitialContext();
var snName = "java:comp/env/mail/MyMailSession";
var session = (jakarta.mail.Session)ic.lookup(snName);
```

3. Override the Jakarta Mail session properties if necessary. For example:

```
var props = session.getProperties();
props.put("mail.from", "user2@mailserver.com");
```

4. Get a `Store` object from the `Session`, then connect to the mail server using the `Store` object's `connect` method. You must supply a mail server name, a mail username, and a password:

```
var store = session.getStore();
store.connect("MailServer", "MailUser", "secret");
```

5. Retrieve the `INBOX` folder.

```
Folder folder = store.getFolder("INBOX");
```

6. It is efficient to read the `Message` objects (which represent messages on the server)

into an array or a collection.

```
Message[] messages = folder.getMessages();
```

Using Application-Scoped Jakarta Mail Resources

You can define an application-scoped Jakarta Mail or other resource for an enterprise application, web module, EJB module, connector module, or application client module by supplying a `payara-resources.xml` deployment descriptor file.

For more details, see "application-scoped-resources" in the Payara Server Application Deployment section.

Using the Data Grid in Your Applications

This section details how to use the underlying Data Grid Hazelcast instance in Payara Server within application code.

Accessing the Hazelcast Instance via JNDI

By default, the JNDI name of the hazelcast instance is `payara/Hazelcast`, though this can be altered using the "set-hazelcast-configuration" command. You will need to import the following packages into your Java class:

```
import com.hazelcast.core.HazelcastInstance;
import javax.naming.Context;
import javax.naming.InitialContext;
```

To import the requisite Hazelcast classes, you will need to set add the Public API artefact as a dependency in the project's `pom.xml` file (in case of Maven projects), or have the Hazelcast JAR set as a project dependency (if using a non-Maven based project).

Here's the Maven dependency:

```
<dependency>
  <groupId>fish.payara.api</groupId>
  <artifactId>payara-api</artifactId>
  <version>${release-id}</version>
```

```
</dependency>
```

The Hazelcast JAR (for non-Maven projects), can either be downloaded from the Hazelcast website, or you can make use of the JAR that is packaged in Payara Server. The JAR packaged with Payara can be found at `<Payara Installation Directory>/glassfish/modules/hazelcast.jar`.

The following code sample will initialize a `HazelcastInstance` variable in the instance embedded in Payara:

```
public class Component{

    public void businessMethod(){
        var ctx = new InitialContext();
        HazelcastInstance instance = (HazelcastInstance) ctx.lookup
("payara/Hazelcast");
    }
}
```

WARNING It is recommended to wrap this block in a `try-catch` clause, since a `NamingException` exception can be generated due to the use of JNDI.

Using Hazelcast for Web and EJB Container Persistence

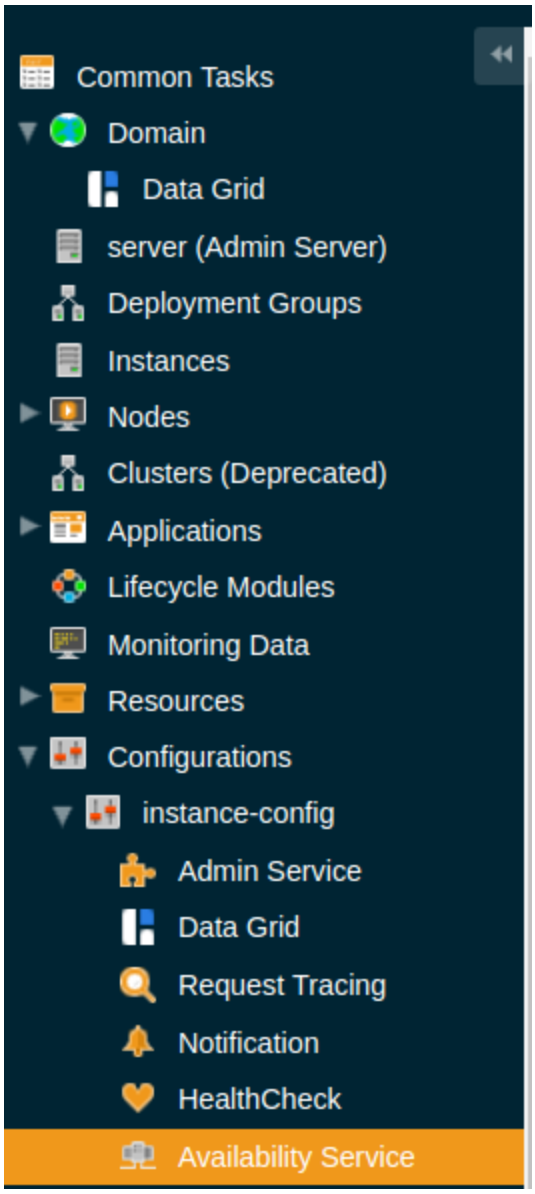
Hazelcast can be used as the persistence provider for both the Web and EJB containers.

Setting Hazelcast as the Persistence Provider through the Admin Console

NOTE Payara Server's initial configuration has the `hazelcast` option as the default setting.

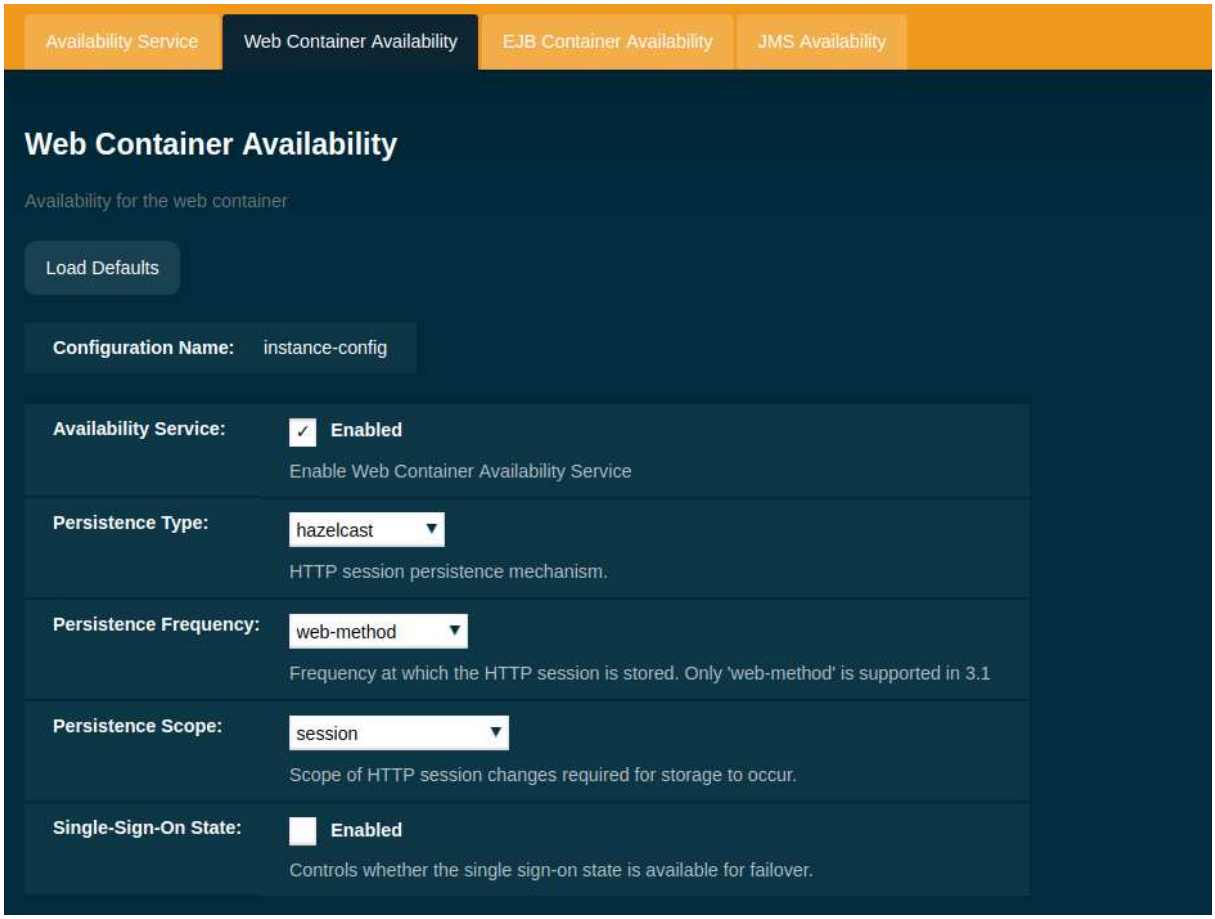
To set up Hazelcast for persistence:

1. Select the instance's configuration from the page tree:
2. Select *"Availability Service"* to view the *"Availability Service"* page:



Setting Web Persistence

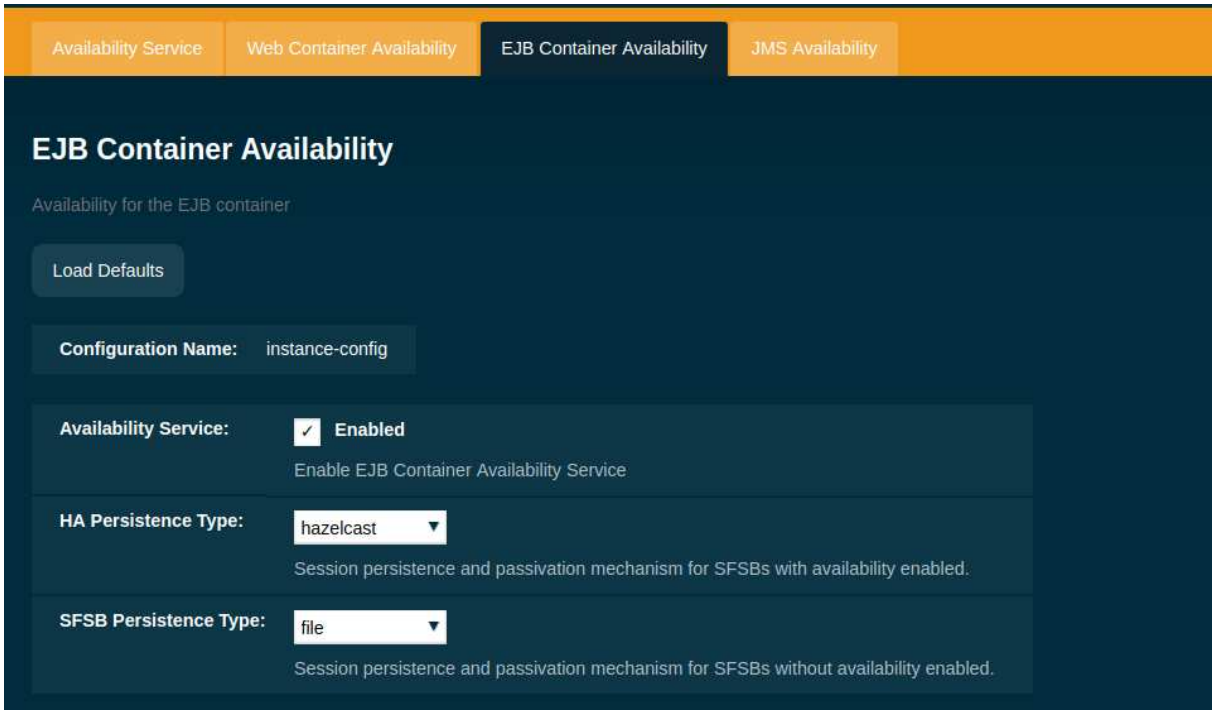
1. Open the "Web Container Availability" tab, and select "Hazelcast" from the *Persistence Type* drop-down menu:



2. Save the changes.

Setting EJB Persistence

1. Open the "EJB Container Availability" tab and select "Hazelcast" from the *Persistence Type* drop-down menu:



2. Save the changes.

Setting Hazelcast as the Persistence Provider from the Command Line

To configure the persistence provider with the *asadmin* CLI, you have to use the *set* command.

For the Web Container

To set Hazelcast as the persistence provider of the Web Container, run the following command:

```
asadmin> set <cluster-config>.availability-service.web-container-
availability.persistence-type=hazelcast
```

For the EJB Container

To set Hazelcast as the persistence provider of the EJB Container, run the following command:

```
asadmin> set <cluster-config>.availability-service.ejb-container-availability.sfsb-
ha-persistence-type=hazelcast
```

Using the Jcache API

This chapter covers how to use the JCache functionality in Azul Payara.

NOTE	*JSR107 (JCache)* is implemented in Payara Server by Hazelcast Community Edition.

IMPORTANT	JCache is not an official specification of the Jakarta Platform and any of its variant profiles. Its inclusion in Azul Payara is a product choice to leverage the capabilities of a caching API as the Jakarta Platform lacks an official API.

Accessing the Caching Provider and Cache Manager

To create a `Cache` instance you will need to access either a `CachingProvider` or a `CacheManager` instance at runtime. This can be done either declaratively via CDI injection or programmatically via JNDI lookup.

Using Injection

You can inject both `CachingProvider` and `CacheManager` instances in any CDI beans like this:

```
import javax.cache.CacheManager;
import javax.cache.spi.CachingProvider;
import jakarta.inject.Inject;

@ApplicationScoped
public class Component{

    @Inject
    CacheManager manager;
    @Inject
    CachingProvider provider;
}
```

NOTE	Your WAR or JAR artifact must be an implicit or explicit bean archive.

Using JNDI

Both the `CachingProvider` and the `CachingManager` components can also be

programmatically accessed through their JNDI names as well:

```
import javax.cache.spi.CachingProvider;
import javax.cache.CacheManager;

public class Component{

    public void businessMethod(){
        Context ctx = new InitialContext();
        CachingProvider provider = (CachingProvider) ctx.lookup
("payara/CachingProvider");
        CacheManager manager = (CacheManager) ctx.lookup("payara/CacheManager");
    }
}
```

Creating a Cache Instance

You can create a cache using either the `CacheManager.getCache` method, or by directly injecting the cache instance into a CDI bean.

Using Injection

Injecting a cache into a CDI bean is simple enough:

```
import jakarta.inject.Inject;
import javax.cache.Cache;

@ApplicationScoped
public class Component{

    @Inject
    Cache cache;
}
```

The name of this cache will be the canonical name of the class it is created in.

NOTE	Caches created in this way will also have JMX statistics and management enabled.

Typed Cache Injection

A typed cache can also be injected in the same manner:

```
import jakarta.inject.Inject;
import javax.cache.Cache;

@ApplicationScoped
public class Component{
```

```
    @Inject
    Cache<Long, Property> cache;
}
```

NOTE	Both key and value types must be `Serializable` so that the cache can be properly instantiated at runtime.

Injecting a Custom Cache

You can determine the name and other attributes of a cache created through injection using the `@NamedCache` annotation.

For example, to inject a cache with a custom name and with JMX management enabled:

```
import fish.payara.cdi.jsr107.impl.NamedCache;
import jakarta.inject.Inject;
import javax.cache.Cache;

@ApplicationScoped
public class Component{

    @NamedCache(cacheName = "custom", managementEnabled = true)
    @Inject
    Cache cache;
}
```

IMPORTANT	This annotation is part of the "Azul Payara Public API"

If you only want to set the name of the cache but don't want to depend on the `@NamedCache`, you can use the `@CacheDefaults` annotation on the bean class instead:

```
import jakarta.inject.Inject;
import javax.cache.Cache;
import javax.cache.annotation.CacheDefaults;
import jakarta.enterprise.context.ApplicationScoped;

@ApplicationScoped
@CacheDefaults(cacheName = "custom")
public class CacheBean {

    @Inject
    Cache cache;
}
```

TIP

Keep in mind that this solution only works if your bean has one injected cache only. If you are in a situation where you must inject more than one cache into the bean then consider using the `@NamedCache` annotation to avoid name collisions.

Using Jcache Annotations

Azul Payara supports cache definitions and operations on caches by using the standard set of annotations that are part of the JCache API:

`@CachePut`: Puts the specified key and value in the cache. **`@CacheRemove`**: Removes an element from the cache that corresponds to the supplied key. **`@CacheDefaults`**: Allows the configuration of defaults for `CacheResult`, `CachePut`, `CacheRemove`, and `CacheRemoveAll` at the class level. **`@CacheKey`**: Marks a method parameter as the key of a cache. **`@CacheValue`**: Marks a method parameter as the value of a cache key.

See Also

- JCache API Javadoc

Using Request Tracing in Applications

Request Tracing in Azul Payara is done either automatically, by using the MicroProfile OpenTracing `@Traced` annotation or by utilizing OpenTelemetry/OpenTracing APIs. This chapter covers manually decorating your applications to enable explicit Request Tracing.

Terminology

Traces & Spans

A Span is a named, timed operation that represents an individual unit of work, such as the execution of a JAX-RS method. Since a Span is a unit of work, they typically contain references to other Spans to represent the work in its entirety.

A Trace is the term used to describe the collection of Spans that make up an entire request's lifecycle from the viewpoint of Azul Payara: from the request hitting the server to it leaving. A Trace can be made up of simply one Span.

References

The references between Spans are of one of two types: "Child of" or "Follows from":

- *"Child of"* references are used to denote that a Span is the child of another Span, e.g. a sub-unit of work.
- *"Follows from"* references are used to denote that a Span was spawned by another Span, but is not a subunit of work of the parent Span (the parent Span does not depend on the child Span).

Tags and Logs

A Span Tag is simply a `key:value` pair that is added to a Span to provide details about the unit of work. For example, JAX-RS methods automatically set the `http.url` tag to provide information about the URL traced.

A Span Log is a Tag that is paired with a timestamp. As denoted by the name, this is typically used to add information to a Span about an event happening at a specific time during the duration of the Span (e.g. an error).

Using the `@Traced` Annotation

The `@Traced` annotation can be placed on CDI Bean methods to automatically trace each call of the method. You can also place the annotation on the class itself to trace all methods within the class, rather than adding the annotation to each method individually.

The `@Traced` annotation accepts two optional parameters:

`operationName`: defines the operation name of the span. If left blank it will default to the canonical method signature. **`value`**: a boolean true or false to determine if the method should be traced. Defaults to true.

```
import org.eclipse.microprofile.opentracing.Traced;

@RequestScoped
public class TracedExample {

    @Traced
    public String method1() {
        return "anon1";
    }
```

```
    @Traced(operationName = "tracingTest")
    public String method2() {
        return "anon2";
    }

}
```

The `@Traced` the annotation operates as an Interceptor, so for the decorated methods of your CDI Beans to be traced they need to be a) not static or private, and b) accessed externally or via self-injection. This means that in the following example, `method1` of the above class would be traced, and `method2` would not:

```
@Path("/")
@RequestScoped
public class JaxrsEndpoints {

    @Inject
    TracedExample tracedExample;

    @GET
    public String method1Traced() {
        return tracedExample.method1();
    }

    @GET
    public String method2NotTraced() {
        return new TracedExample().method2();
    }

}
```

TIP Annotations applied to methods take precedence over those applied to classes. This allows you to whitelist or blacklist methods for tracing.

The Maven dependency to add to your application to get access to the `@Traced` annotation is the following:

```
<dependency>
    <groupId>org.eclipse.microprofile.opentracing</groupId>
    <artifactId>microprofile-opentracing-api</artifactId>
    <version>{mpOpenTracingVersion}</version>
    <scope>provided</scope>
</dependency>
```

Using the Opentracing `Tracer` Class

The OpenTracing `Tracer` class allows you to manually start, finish, and add information to Spans.

To access the tracer in your applications, you can inject it like so:

```
import io.opentracing.Tracer;

public class TracedComponent{

    @Inject
    Tracer tracer;
}
```

CAUTION	You can only inject a Tracer when your application is running inside an Azul Payara runtime you cannot do so from a Java SE client. From a Java SE client you must use the OpenTracing.io `GlobalTracer` class, see getting-a-tracer for more details.

Starting a Span, marking it as the active Span (the one from which any children will spawn), and ending it is done like so with a *try-with-resources* block:

```
import io.opentracing.Tracer;

public class TracedComponent{
    public void tracerExample() {
        try (Span span = tracer.buildSpan("example").startActive(true)) {
            // Code to trace
        }
    }
}
```

If you wish to add any Tags or Logs to your Spans, even those started by the `@Traced` annotation, you can do this like so:

```
import io.opentracing.Tracer;

@Path
@RequestScoped
public class TracedResource{

    @Inject
    Tracer tracer;
```

```
    @GET
    @Path
    @Traced
    public String example2() {
        tracer.activeSpan().setTag("exampleTag", "foo").log("exampleLog");
        return "anon";
    }
}
```

The `Tracer` class also gives you access to the `inject` and `extract` methods, which you can use for tracing across threads and processes. It is recommended however that you simply use a Managed Executor Service for your asynchronous tasks and the default JAX-RS client implementation or MicroProfile REST Client for JAX-RS calls as these will automatically do this for you.

Disabling Automatic Tracing of JAX-RS Methods and Microprofile REST Client Calls

By default, calls to JAX-RS methods and any calls by a MicroProfile REST Client are traced.

Disabling Automatic Tracing of JAX-RS Methods

To disable tracing of JAX-RS methods, you can use the `@Traced` annotation on each method or class that you wish to skip tracing of.

```
import org.eclipse.microprofile.opentracing.Traced;

@Path("/")
@RequestScoped
public class JaxrsEndpoints {

    @Inject
    TracedExample tracedExample;

    // Automatically traced
    @GET
    public String method1Traced() {
        return tracedExample.method1();
    }

    // Tracing disabled
    @GET
    @Traced(false)
    public String method2NotTraced() {
        return new TracedExample().method2();
    }
}
```

You can also use tracing of JAX-RS methods by defining a skip pattern using a MicroProfile Config properties file or config source. These skip patterns accept Java regular expressions to match against the URI of your JAX-RS methods.

```
mp.opentracing.server.skip-pattern=/foo|/bar.*
```

Disabling Automatic Tracing of Microprofile REST Client

To disable tracing of MicroProfile REST Client calls, annotate the client interface or method with `@Traced(false)`.

```
import org.eclipse.microprofile.opentracing.Traced;

@Traced(false)
@Path("/")
public interface ExampleClient {

    @GET
    @Path("test")
    @Produces(MediaType.TEXT_PLAIN)
    Response example();
}
```

Tracing APIs Compatibility Matrix

The following is an up-to-date compatibility matrix that list the compatible versions of OpenTelemetry and OpenTracing in Azul Payara:

Payara Server & Micro Version	OpenTracing.io API Version	OpenTelemetry API Version
6.0.0 - 7.2.0	0.33	1.29.0
5.36.0+	0.33	-
4.1.2.183 - 4.1.2.191 5.183 - 5.35.0	0.31	-
4.1.2.182 5.182	0.30	-

Features Not Currently Supported

Azul Payara currently does not offer the following OpenTracing features:

- Conversion to or from OpenTracing `0.31` or newer APIs

Public API

Public API

Public API

This page contains information on using the Public API integrated within Payara Server and Payara Micro.

> **TIP** | See the Payara API Javadoc for the complete public Payara API reference.

Use Cases

The Public API should be used in the following scenarios:

1. Java EE Developers and end users that develop web and enterprise applications that require Payara specific features outside the specifications (and **JCache**).
2. Contributors that want to develop extensions for Payara Server or Payara Micro.

Maven Coordinates

The Public API is released as a Maven artifact in **Payara Nexus**, in the `payara-artifacts` repository. To use it in a Maven project, set the following dependencies in your project's POM:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>fish.payara.api</groupId>
            <artifactId>payara-bom</artifactId>
            <version>7.2.0</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>fish.payara.api</groupId>
        <artifactId>payara-api</artifactId>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

For information on the Payara BOM, please see "Maven Bom".

Firing and Listening for Remote CDI Events

Payara Server is able to listen to and fire CDI events across the Hazelcast cluster. The CDI events can be fired and listened to by applications on Payara Server instances as well as Payara Micro instances provided that they are connected to the same cluster.

Requirements

Hazelcast must be enabled for all Payara Server instances and they must be able to connect to the same Hazelcast cluster. See the "Data Grid" section for more details.

Initializing the Event Bus

The event bus is initialized automatically after an application is started. No additional configuration is necessary to fire and listen to remote events.

Firing Remote CDI Events

In order to fire a CDI event remotely, mark it with the `@Outbound` annotation:

```
@Inject
@fish.payara.micro.cdi.Outbound
Event<String> event;
```

The above will create an event with a String payload, which will be observable on remote instances in the cluster. An outbound event can be fired as any other CDI event:

```
event.fire("This is an event");
```

The payload passed as an argument to the `fire()` method has to implement the `Serializable` interface.

An outbound event will be fired asynchronously on a remote instance. If loopback is allowed, it will be fired also on the same instance, but in a different thread. Therefore the call doesn't block the current thread.

TIP	It's possible to use additional qualifiers to match event with observers. Any

> additional qualifier used with `@Outbound` will mean that the event will be matched with an `@Inbound` listener that accepts the qualifiers.

Observing Remote CDI Events

Outbound events are observed by every CDI bean with a listener method that is marked with the `@Inbound` qualifier:

```
public void receiveEvent(@Observes @Inbound String message) {
}
```

The observers may reside in any application running on one or more instances in a Hazelcast cluster, and they will listen to events fired in an application running on any instance within the same cluster. By default, events will be fired on all remote cluster instances (all instances except the one in which the event was fired).

An event will be matched with observers by the payload class, using the same rules as usual CDI events. Observers will receive events that are assignable to the observed class.

Additional Options

Using CDI Qualifiers

It's possible to tag a remote event in a type-safe way using any CDI qualifier. Only observers that match the qualifiers would receive such event.

A remote event with a qualifier `@MyQualifier` can be triggered like this:

```
@Inject
@fish.payara.micro.cdi.Outbound
@MyQualifier
Event<String> event;
...
event.fire("This is a qualified event");
```

This event can be observed only by an observer that contains the qualifier:

```
public void receiveEvent(@Observes @Inbound @MyQualifier String message) {
}
```

Event Name

It is possible to tag a remote event with a name. In that case, only observers that listen to events of the same name will receive such events. To fire an event with a name:

```
@Inject
@Outbound(eventName = "MyEvent")
Event<String> event;
```

To receive the above named event:

```
public void receiveEvent(@Observes @Inbound("MyEvent") String message) {
}
```

Instance Name

By default, all remote instances in the same cluster will receive an outbound event, which matches the observed class and event name. If you want to target specific instances in the cluster, it is possible to define one or more target instances for an outbound event:

```
@Inject
@Outbound(instanceName = "Instance-1")
Event<String> event;
```

Several instance names can be configured with an array:

```
@Inject
@Outbound(instanceName = {"Instance-1", "Instance-2"})
Event<String> event;
```

If you are using Payara Server, the name of the DAS instance is `server` by default. In the case of a separate standalone instance, use the instance's name in the configuration. If you are using Payara Micro, the instance's name is configured using the `--name` option, or is generated by if not configured (see the section "Payara Micro Instance Names").

Loopback

When an outbound event is fired, by default, it is treated as a remote message and will

be delivered only to remote instances in the cluster. It will not be delivered to listeners on the same server instance.

If an event should be delivered to observers in the local instance as well, it can be done by setting the `loopBack` attribute to true:

```
@Inject
@Outbound(loopBack = true)
Event<String> event;
```

Dynamic Configuration

If required, events can also be fired programmatically, not only using the static `@Outbound` annotation next to the injection point. Although there is no special API to support it, CDI already supports programmatic configuration of events.

The following is an example of how to configure and fire an outbound event programmatically without using any injection points:

```
CDI.current().getBeanManager().fireEvent("This is an event", new Outbound() {
    @Override
    public String eventName() {
        return System.getProperty("eventname");
    }

    @Override
    public boolean loopBack() {
        return Boolean.getBoolean(System.getProperty("loopback"));
    }

    @Override
    public String instanceName() {
        return System.getProperty("instanceName");
    }

    @Override
    public Class<? extends Annotation> annotationType() {
        return this.getClass();
    }
});
```

The above code creates a dynamic instance of the `@Outbound` annotation and fires the event `"This is an event"` using a bean manager retrieve via a static `CDI.current`()` method.

Clustered Singleton

The Payara API provides a `@Clustered` annotation that makes `@ApplicationScoped` CDI beans or `@Singleton` EJB beans cluster-wide. This allows a single bean to be shared across an entire cluster.

How It Works

The singleton bean is deployed and will be available to all members of the cluster. Prior to any call to this bean, its latest version will be retrieved from a Hazelcast map, which will be possibly updated after the call is finished. In order to avoid race conditions or data corruption, a **Hazelcast distributed lock** for the bean will be optionally acquired so that only one method call can be executed in the cluster at a time.

Examples

Examples can be found here: https://github.com/payara/Payara-Examples/tree/master/javaee/clustered-singleton

`@Clustered` and EJB Timers

Persistent EJB Timers (which are the default) will work correctly with clustered singletons since they will only be executed in a single node of the cluster or deployment group.

IMPORTANT	If the purpose of a clustered singleton bean is to only define a persistent EJB timer which needs to be fired **once** per cluster or deployment group on a Payara Server domain, then it is better to use a stateless session bean instead. The default behaviour of Payara Server is to trigger expired persistence EJB timers once per deployment group or cluster, so the added redundancy provided by this feature is not necessary.

`@Clustered` and `@Startup` / `@Initialized`/ `@ApplicationScoped`

Care must be taken in this scenario. By default, the `@PostConstruct` lifecycle method is called for every instance in the cluster for these beans. This behavior needs to be disabled by using `@Clustered(callPostConstructOnAttach = false)` and

possibly `callPreDestroyOnDetach = false` as well, so only the `@PostConstruct` method will be called only once per cluster. `@Observed @Initialized` methods are called for every instance in the cluster. The only valid method for cluster-wide initialization is to use the `@PostConstruct` lifecycle method configured as described above.

CDI Considerations and Notes

Since there are no default locks for CDI, `@Clustered` CDI `@ApplicationScoped` beans do not have distributed locks on by default. It is highly recommended to enable distributed locks for CDI Clustered beans: `@Clustered(lock = DistributedLockType.LOCK)` so there is no chance of race condition / data corruption for CDI Clustered beans.

Requirements and Assumption

1. Hazelcast is required to be enabled (default)
2. `@Clustered` only works with `@ApplicationScoped` or `@Singleton` annotation, and in no other circumstance
3. All fields in the `@Clustered @Singleton` class need to be correctly Serializable
4. Works correctly only if accessed via @Injected proxy, not through direct access
5. Works with EJB timers correctly only if all timers are persistent (default)
6. All method calls to `@Clustered` beans operate on a local instance
7. Data corruption is possible if distributed lock is turned off and there is a race condition between method calls to the same bean on multiple cluster nodes simultaneously. In such a case, last method call wins the singleton state
8. Clustered singletons are stored in Hazelcast map under "Payara/(EJB/CDI)/singleton/<singletonComponentName>/[lock]"

IMPORTANT	It is also highly recommended to add a `serialVersionUID` variable to the `@Clustered @Singleton` class, in order to verify that a loaded class and the serialized object are compatible.

> A previously serialized entry in the Data Grid is compatible with an updated version of the same class.

```
private static final long serialVersionUID = 1234L;
```

Adding a `serialVersionUID` is expected to prevent serialization issues encountered by the Data Grid if your singleton's source code is updated regularly.

Usage

A singleton bean is made cluster-wide by annotating the class with the `@fish.payara.cluster.Clustered` qualifier as well as it's scope annotation.

Alternatively, you can use the `payara-ejb-jar.xml` to configure a Singleton EJB to be cluster-wide.

Example

An example for a CDI bean:

```
@Clustered
@jakarta.enterprise.context.ApplicationScoped
public class ClusterSingletonBean implements Serializable {

}
```

An example for a Singleton EJB:

```
@Clustered
@jakarta.ejb.Singleton
public class ClusterSingletonBean implements Serializable {

}
```

An example for a Singleton EJB using `payara-ejb.jar` (`clustered-key-name` is optional):

```
<payara-ejb-jar>
    <enterprise-beans>
        <ejb>
            <ejb-name>ClusteredSingleton</ejb-name>
            <clustered-bean>true</clustered-bean>
            <clustered-key-name>ClusteredSingleton</clustered-key-name>
```

```
        </ejb>
    </enterprise-beans>
</payara-ejb-jar>
```

Configuration

The `@Clustered` annotation has several configuration options. They are detailed below.

Table 197. Configuration Options

Option	XML element	Description	Default
keyName	clustered-key-name	The key in the distributed map to bind the clustered object to.	The name of the bean. If the bean is a CDI bean and has no assigned name, the bean class name will be used.
lock	clustered-lock-type	The type of distributed locking to be performed. For EJB beans, only `INHERIT` and `LOCK_NONE` are valid. For CDI beans, valid values are `LOCK` and `INHERIT`, which is equivalent to using `LOCK_NONE`.	`INHERIT`
callPostConstructOnAttach	`clustered-attach-postconstruct`	Whether to call `@PostConstruct` each time the bean is created on a different node. Will result in multiple calls.	`true`
callPreDestroyOnDetach	`clustered-detach-predestroy`	Whether to call `@PreDestroy` when the singleton is destroyed on an instance while still being available on another. Will result in multiple calls.	`true`

Distributed Locking

Clustered singleton beans allow a locking type, to specify how the distributed object is locked when being accessed by multiple instances. The lock options are members of the class `fish.payara.cluster.DistributedLockType`, which are as follows:

- `LOCK` - Distributed locking will be performed.
- `LOCK_NONE` - No distributed locking will be performed.
- `INHERIT` - The locking behaviour will be inherited from the inherited class.

By default, `@Singleton` EJBs will use a distributed lock, and `@ApplicationScoped` CDI beans won't.

When a distributed object is locked, it will only be written by one thread across the entire cluster at any one time. Locks use system resources, but prevent synchronisation errors with the singleton data.

NOTE	If a member holding a lock goes offline, the lock will become available again.

Transactions

Transactions in a clustered singleton work the same way that they would work in EJB or CDI depending on which scope annotation you're using. Transactions are not distributed through the whole cluster. When a transaction is created in a thread in one JVM, it must be handled and closed in the same thread; it cannot be passed onto a different server instance. Once the transaction is closed, the changes will be replicated to the rest of the cluster.

OAuth2 Support

The Payara API provides a `@OAuth2AuthorisationMechanism` annotation that creates an authorisation mechanism for OAuth2 support. This works in the same way as other authorisation mechanisms in the Java EE Security API.

Usage

The OAuth2 authentication mechanism is defined through the `@OAuth2AuthorisationMechanism` annotation. Specifying this in a valid place as defined by the Java EE Security API will create the mechanism. Often this may mean that any class is a valid position.

Example

Here's an example that configures a Google OAuth2 endpoint:

```
@OAuth2AuthenticationDefinition(
    authEndpoint="https://accounts.google.com/o/oauth2/v2/auth",
    tokenEndpoint="https://www.googleapis.com/oauth2/v4/token",
    clientId="{my-key}.apps.googleusercontent.com",
    clientSecret="{my-secret}",
    redirectURI="https://localhost:8181/{my-application}/callback",
    scope="email",
    extraParameters = {
        "testKey=testValue",
        "testKey2=testValue2"
    }
)
public class SecurityBean {

}
```

IMPORTANT	Once the user is authenticated they will not have any roles defined, regardless of the OAuth scope. Roles should be specified using `@DeclareRoles`. See this sample project for a more detailed example.

When defining a OAuth2 flow within an application deployed on Payara Server it is possible to retrieve the access token within any bean in the scope of the callback/redirectURI resource used to configure the authentication:

```
@WebServlet("/callback")
public class CallbackServlet extends HttpServlet {

    @Inject
    OAuth2AccessToken token;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        //Here's the access token
        out.println(token.getAccessToken());
    }
}
```

IMPORTANT	OAuth2 authorization support require the use of HTTP/2, so make sure to use the appropriate HTTP network listener when preparing

> your application.

Configuration

The `@OAuth2AuthenticationDefinition` annotation has several configuration options.

They are detailed below.

Table 198. Configuration Options

Option	Required	Description	Requirements
`authEndpoint`	true	The URL for the OAuth2 provider to provide authentication.	The endpoint must be HTTPS.
`tokenEndpoint`	true	The URL for the OAuth2 provider to give the authorisation token.	The endpoint must be HTTPS.
`clientId`	true	The client identifier issued when the application was registered.	N/A.
`clientSecret`	true	The client secret for the registered application.	N/A.
`scope`	false	The scopes requested from the OAuth provider.	N/A.
`redirectURI`	false	The URL to redirect the user to upon successful authentication.	Must be equal to one set in the OAuth2 Authentication provider.
`extraParameters`	false	An array of extra options to be sent to the OAuth provider.	Must be in the form `"key=value"`.

Expression Language Support

Additionally, the `@OAuth2AuthenticationDefinition` supports the use of expression language (EL) notation for dynamic configuration scenarios. This means that you can use any CDI bean properties to set the OAuth2 configuration like this:

```
@OAuth2AuthenticationDefinition(
    authEndpoint="#{oauth2ConfigBean.authEndpointURL}",
    tokenEndpoint="#{oauth2ConfigBean.tokenEndpointURL}",
    clientId="#{oauth2ConfigBean.clientId}",
```

```
    clientSecret="#{oauth2ConfigBean.clientSecret}",
    redirectURI="#{oauth2ConfigBean.redirectURI}"
)
public class SecurityBean {
}
```

Client Secret Aliasing

The client secret can be input directly, or for added security it can be aliased using any of the following features:

- administering-password-aliases
- "Environment Variables / System Properties"
- "Config API"

See Also

- OAuth v2 Official site

Openid Connect Support

The Payara API provides a `@OpenIdAuthenticationDefinition` annotation that creates an authorization mechanism for OpenID Connect support. This works in the same way as other authorization mechanisms in the Java EE Security API.

Usage

The OpenID Connect authentication mechanism is defined through the `@OpenIdAuthenticationDefinition` annotation. Specifying this in a valid place as defined by the Java EE Security API will create the mechanism.

Example

Here's an example that configures a OpenID Connect client:

```
@OpenIdAuthenticationDefinition(
       providerURI = "https://sample-openid-server.com",
       clientId = "87068hgfg5675htfv6mrucov57bknst.apps.sample.com",
       clientSecret = "{my-secret}",
       redirectURI = "${baseURL}/callback",
       extraParameters = {
            "testKey=testValue",
            "testKey2=testValue2"
       }
)
```

```
public class SecurityBean {

}
```

See this sample project for a more detailed example.

When defining a OpenID Connect flow within an application deployed on Payara Server, it is possible to retrieve the access token, identity token, user claims and the other authentication information within any bean in the scope of the callback/redirectURI resource used to configure the authentication:

```
@WebServlet("/callback")
public class CallbackServlet extends HttpServlet {

    @Inject
    OpenIdContext context;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        //Here's the caller name
        out.println(context.getCallerName());
        //Here's the caller groups
        out.println(context.getCallerGroups());
        //Here's the unique subject identifier within the issuer
        out.println(context.getSubject());
        //Here's the access token
        out.println(context.getAccessToken());
        //Here's the identity token
        out.println(context.getIdentityToken());
        //Here's the user claims
        out.println(context.getClaimsJson());
    }
}
```

The original protected URL called is stored so that a redirect can be issued to this page after the callback from the OpenIdConnect provider is handled.

NOTE Only some basic redirect is supported namely a GET. Only the URL and query parameters are available, not the headers and the request body.

```
@WebServlet("/callback")
public class CallbackServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
        // handling the OpenIdContext information. See previous example
        // Now return to original requested page
        response.sendRedirect(request.getSession().getAttribute(OpenIdConstant
.ORIGINAL_REQUEST).toString());
    }
}
```

Configuration

OpenID Client can be configured with both `@OpenIdAuthenticationDefinition` annotation attributes and Microprofile Config properties. The annotation and MicroProfile properties has several configuration options.

CAUTION

If you are using MicroProfile Config properties and the value contains a placeholder that should not be resolved by MicroProfile Config itself but by the Server (like `payara.security.openid.redirectURI=${baseURL}/Callback`) make sure the value is properly escaped. It will otherwise result in `NoSuchElementException`.
Here's an example how such a value should be escaped: `payara.security.openid.redirectURI=\\${baseURL}/Callback`

The annotation and MicroProfile properties has several configuration options. They are detailed below.

Table 199. Configuration Options for `@OpenIdAuthenticationDefinition`

Option	MP Config property	Required/Type	Description	Default value	Requirements
`providerURI`	`payara.security.openid.providerURI`	`false`	The provider URI to discover the metadata of the OpenID Connect provider. If not specified, metadata are read from `providerMetadata`.		The endpoint must be HTTPS.

Option	MP Config property	Required/Type	Description	Default value	Requirements
`providerMetadata`		required if `providerURI` not specified, type: `@OpenIdProviderMetadata`	Alternative way to specify metadata if auto-configuration from `providerURI` can't be used.	`payara.security.openid.provider.*`	The endpoint must be HTTPS.
`clientId`	`payara.security.openid.clientId`	`true`	The client identifier issued when the application was registered.		N/A.
`clientSecret`	`payara.security.openid.clientSecret`	`true`	The client secret for the registered application.		N/A.
`redirectURI`	`payara.security.openid.redirectURI`	`true`	The URL to redirect the user to upon successful authentication. Can include the variable `${baseURL}`, which is replaced by the URL of the application's context path (e.g. `https://myhost:8080/myapp`)	`${baseURL}/Callback`	Must be equal to one set in the OpenID Connect provider.
`scope`	`payara.security.openid.scope`	`false`	The scopes requested from the OpenID Connect provider.	{"openid", "email", "profile"}	N/A.

Option	MP Config property	Required/Type	Description	Default value	Requirements
`responseType`	`payara.security.openid.responseType`	`false`	Response Type value defines the processing flow to be used.	code	N/A.
`responseMode`	`payara.security.openid.responseMode`	`false`	Informs the Authorization Server of the mechanism to be used for returning parameters from the Authorization Endpoint.		N/A.
`prompt`	`payara.security.openid.prompt`	`false`	The prompt value specifies whether the authorization server prompts the user for re-authentication and consent.		N/A.
`display`	`payara.security.openid.display`	`false`	The display value specifying how the authorization server displays the authentication and consent user interface pages.	page	N/A.
`useNonce`	`payara.security.openid.useNonce`	`false`	Enables string value used to mitigate replay attacks.	`true`	N/A.
`useSession`	`payara.security.openid.useSession`	`false`	If enabled state & nonce value stored in session otherwise in cookies.	`true`	N/A.

Option	MP Config property	Required/Type	Description	Default value	Requirements
`jwksConnectTimeout`	`payara.security.openid.jwks.connect.timeout`	`false`	Sets the connect timeout(in milliseconds) for remote JWKS retrieval.	`500`	Value must not be negative and if value is zero then infinite timeout.
`jwksReadTimeout`	`payara.security.openid.jwks.read.timeout`	`false`	Sets the read timeout(in milliseconds) for remote JWKS retrieval.	500	Value must not be negative and if value is zero then infinite timeout.
`tokenAutoRefresh`	`payara.security.openid.token.autoRefresh`	`false`	Enables or disables the automatically performed refresh of Access and Refresh Token.	false	N/A.
`tokenMinValidity`	`payara.security.openid.token.minValidity`	`false`	Sets the minimum validity time(in milliseconds) the Access Token must be valid before it is considered expired.	`10000`	Value must not be negative.

Option	MP Config property	Required/Type	Description	Default value	Requirements
`claimsDefinition.callerNameClaim`	`payara.security.openid.callerNameClaim`	`false`	Defines the name of callerName claim and maps the claim's value to caller name value in IdentityStore#validate.	preferred_username	N/A.
`claimsDefinition.callerGroupsClaim`	`payara.security.openid.callerGroupsClaim`	`false`	Defines the name of callerGroups claim and maps the claim's value to caller groups value in IdentityStore#validate.	groups	N/A.
`proxyDefinition.hostName`	`payara.security.openid.proxyHostname`	`false`	Defines the hostname of the proxied server.		N/A.
`proxyDefinition.port`	`payara.security.openid.proxyPort`	`false`	Defines the port of the proxied server.		N/A.

Option	MP Config property	Required/Type	Description	Default value	Requirements
`extraParameters`		`false`	An array of extra options to be sent to the OpenID Connect provider.		Must be in the form `key=value` and follow URL query pattern, i.e. `key1=value1&key2=value2&key2=value+with+spaces`. Key entries can be repeated
`logout`		@LogoutDefinition	Defines the functionality that is performed when the user logs out and defines the RP Session Management configuration.		

NOTE	If both an annotation attribute and a MicroProfile Config property are defined for the same option then the MicroProfile Config property value

always takes precedence over the `@OpenIdAuthenticationDefinition` annotation value.

Table 200. Configuration Options For `@OpenIdProviderMetadata`

Option	MP Config property	Description	Default value
`endSessionEndpoint`	`payara.security.openid.provider.endSessionEndpoint`	OIDC provider's logout endpoint URL. If set, overrides the URL obtained via the `end_session_endpoint` element of the OIDC provider's metadata.	
`disableScopeValidation`	`payara.security.openid.disableScopeValidation`	This property disables the scope validation for custom scope configurations	false

Expression Language Support

The `@OpenIdAuthenticationDefinition` supports the use of expression language (EL) notation for programmatic configuration scenarios. This means that you can use any CDI bean properties to set the OpenID Connect configuration like this:

```
@OpenIdAuthenticationDefinition(
    providerURI="#{openidConfigBean.tokenEndpointURL}",
    clientId="#{openidConfigBean.clientId}",
    clientSecret="#{openidConfigBean.clientSecret}",
    redirectURI="#{openidConfigBean.redirectURI}"
)
public class SecurityBean {
}
```

IMPORTANT	By default, the EL expressions are evaluated only once after the application is loaded and the evaluated values are then remembered until the application is reloaded, for performance reasons. This means that although the configuration can be evaluated dynamically the first time it's needed, it's not possible to change the configuration later on.

> If you need to dynamically modify the configuration during the lifetime of the application, follow the next section about multi-tenancy support.

Multi-Tenancy Support (Session-Scoped Configuration)

By default, the same configuration of the OpenID connector is applied for the whole application, for all authentication attempts. This is for performance reasons. The OpenID connector also supports re-evaluating the configuration for each user session, before each authentication attempt.

This is useful in a multi-tenant scenario to define a different configuration for each tenant. It's also useful if the user should be able to select which provider they want to use to authenticate.

To enable re-evaluation of the configuration for each user session, set the MicroProfile Configuration property `payara.security.openid.sessionScopedConfiguration` to `true`. To specify it directly in the application, you can place it in the microprofile-config.properties file in the `META-INF` directory on the classpath (in a WAR application it could be in `WEB-INF/classes/META-INF`).

With this enabled, it's possible to use EL expressions to dynamically adjust the configuration before each authentication attempt, e.g. based on any information in the incoming HTTP request. The information about the HTTP request can be retrieved from a `HttpServletRequest` object injected using `@Inject`.

> **IMPORTANT** It's not possible to use a different configuration for just a subset of secured resources. Once a user is authenticated, the authentication information is saved in the HTTP session. All secured resources will be accessed using the same user, having the same roles, until the user logs out.

Example of Multi-Tenant Authentication

This example should showcase the following steps:

- Enable session-scoped OpenID Connect configuration
- Resolve the tenant name from an HTTP request query parameter
- Use the tenant name to read the configuration value from the respective "MicroProfile Config" property
- Retrieve the value from an EL expression defined in the `@OpenIdAuthenticationDefinition` annotation

> **TIP**
>
> The tenant can also be resolved from a cookie, which is set the first time a user loads the application; from the domain name in the URL (if different tenants use a different domain name to access the same application); from a path prefix that follows the context root and prepends all application URLs (e.g. `contextroot/tenant1/index.xhtml`, `contextroot/tenant2/index.xhtml`).

1. Create a file `microprofile-config.properties` in your application (for a WAR application it would be in the `WEB-INF/classes/META-INF` directory), with the following contents:

   microprofile-config.properties

   ```
   payara.security.openid.tenant1.providerURI=<TENANT1_OPENID_PROVIDER_URI>
   payara.security.openid.tenant2.providerURI=<TENANT2_OPENID_PROVIDER_URI>
   payara.security.openid.sessionScopedConfiguration=true
   ```

   This will provide configuration for `tenant1` and `tenant2` tenants. For each additional tenant, add a new line for its `providerURI`.

2. Create an `OpenidConfigBean` class with the `tokenEndpointURL` method. This class will be a CDI bean that injects `HttpServletRequest` to get information about which tenant to use. It will also inject `Config` to retrieve the configuration about each tenant from the `microprofile-config.properties` file:

OpenidConfigBeanEL.java

```
@Named
public class OpenidConfigBeanEL {

    @Inject
    HttpServletRequest request;

    @Inject
    Config config;

    private static final String BASE_OPENID_KEY = "payara.security.openid";

    public String getTokenEndpointURL() {
        String tenant = getTenant(request);  // a custom method to decide which
tenant to use
        return config
                .getOptionalValue(BASE_OPENID_KEY + "." + tenant +
".providerURI", String.class)
                // e.g. payara.security.openid.tenant1.providerURI for "tenant1"
tenant
                .orElseGet(() -> {
                    // read config for the "tenant1" tenant by default
                   return config.getValue(BASE_OPENID_KEY +
".tenant1.providerURI", String.class);
                });
    }

    private String getTenant(HttpServletRequest request) {
        return request.getParameter("tenant"); // resolves the tenant name from
a query parameter
    }

}
```

3. Finally, configure the OpenID Connector using the `OpenIdAuthenticationDefinition` annotation that references the `getTokenEndpointURL()` in an EL expression:

   SecurityBean.java

```
@OpenIdAuthenticationDefinition(
        providerURI = "#{openidConfigBean.tokenEndpointURL}",
        clientId = CLIENT_ID_VALUE,
        clientSecret = CLIENT_SECRET_VALUE,
        redirectURI = "${baseURL}/Callback"
)
public class SecurityBean {
}
```

Logout Functionality

With the `logout` parameter of the `OpenIdAuthenticationDefinition` you can

define the behavior when the user logs out of the application and defines how the RP session is managed.

Table 201. Configuration Options

Option	MP Config property	Required	Description	Default value
`notifyProvider`	`payara.security.openid.provider.notify.logout`	`false`	Notify the OIDC provider (OP) that the user has logged out of the application and might want to log out of the OP as well. If `true` then after having logged out the user from RP, redirects the End-User's User Agent to the OP's logout endpoint URL. This URL is normally obtained via the `end_session_endpoint` element of the OP's metadata or can be customized via `fish.payara.security.annotations.OpenIdProviderMetadata#endSessionEndpoint`	`false`
`redirectURI`	`payara.security.openid.logout.redirectURI`	`false`	The post logout redirect URI to which the RP is requesting that the End-User's User Agent be redirected after a logout has been performed. If the redirect URI is empty then redirect to OpenID connect provider *authorization_endpoint* for re-authentication.	

Option	MP Config property	Required	Description	Default value
`accessTokenExpiry`	`payara.security.openid.logout.access.token.expiry`	`false`	Whether the application session times out when the Access Token expires.	`false`
`identityTokenExpiry`	`payara.security.openid.logout.identity.token.expiry`	`false`	Whether the application session times out when the Identity Token expires.	`false`

A programmatic logout is performed by calling `OpenIdContext#logout()` which invalidates the RP's active OpenId Connect session.

If `fish.payara.security.annotations.LogoutDefinition#notifyProvider` is set to true then it redirects the End-User's User Agent to the `end_session_endpoint` to notify the OP that the user has logged out of the RP's application.

It will also ask the user whether they want to log out from the OP as well. After successful logout, the End-User's User Agent redirects back to the RP's `post_redirect_uri` configured via the `fish.payara.security.annotations.LogoutDefinition#redirectURI` property.

Proxy Definition

Using the `proxyDefinition` parameter of the `@OpenIdAuthenticationDefinition` annotation you can specify details regarding proxy mapping within the OpenID Connect client configuration. It contains attributes to specify the `hostName` and `port` of the proxied server. This addition facilitates the configuration of proxy mapping for requests traversing through reverse web proxies.

Moreover, the proxy configuration details (`hostName` and `port`) can be dynamically utilized to define the `redirectURI` within the `OpenIdAuthenticationDefinition`, replacing the `baseURL` placeholder if present. This allows for a more flexible setup,

especially in scenarios involving reverse web proxies.

Scenarios without Proxy

- **Case 1**
  - `redirectURI` employs the `${baseURL}` placeholder, which defaults to `${baseURL}/Callback` when `redirectURI` is not explicitly defined.
  - Proxy definition is also absent.
  - **Outcome:** The generated `redirectURI` should include the request's host and port. The registered Authorized Redirect URI in Google Cloud must contain a URL with the Payara Server instance's host and port.
- **Case 2**
  - `redirectURI` is explicitly defined with the Payara Server instance's host and port.
  - Proxy definition is not present.
  - **Outcome:** The registered Authorized Redirect URI in Google Cloud must contain a URL with the Payara Server instance's host and port.

Scenarios with Proxy

- **Case 3**
  - `redirectURI` employs the `${baseURL}` placeholder, defaulting to `${baseURL}/Callback` when `redirectURI` is not explicitly defined.
  - Proxy definition is included.
  - **Outcome:** The generated `redirectURI` should include the proxy's host and port. The registered Authorized Redirect URI in Google Cloud must contain a URL with the proxy's host and port.
- **Case 4**
  - `redirectURI` is explicitly defined with the Payara Server instance's host and port.

- Proxy definition is also present.
- **Outcome:** The registered Authorized Redirect URI in Google Cloud must contain a URL with the Payara Server instance's host and port.

- **Case 5**
  - `redirectURI` is explicitly defined with the Proxy's host and port.
  - Proxy definition is included.
  - **Outcome:** The registered Authorized Redirect URI in Google Cloud must contain a URL with the proxy's host and port.

Provider Metadata

If the OpenId server doesn't provide autoconfiguration, or it is necessary to customize it, it is possible to set these values in the `providerMetadata` attribute of the `@OpenIdAuthenticationDefinition` annotation.

It's also possible to specify all values by using MicroProfile Config properties. None of the attributes are required in the annotation, but some options are required and must be specified either in the annotation or a MicroProfile property or must be provided by the OIDC provider.

TIP The order of evaluation is: MicroProfile Config → `@OpenIdProviderMetadata` → automatic configuration on `providerURI` address.

When these values, which correspond to lists (e.g. `scopesSupported`, `responseTypesSupported`), are loaded from MicroProfile Config, they are separated by a comma, following MicroProfile Config Array conventions.

Table 202. Provider Metadata Options

Option	MP Config property	Required	Description
`issuer`	`payara.security.openid.provider.issuer`	`true`	The base address of OpenId Connect Provider (OIDC Provider).
`authorizationEndpoint`	`payara.security.openid.provider.authorizationEndpoint`	`true`	The URL for the OAuth2 provider to provide authentication.
`tokenEndpoint`	`payara.security.openid.provider.tokenEndpoint`	`true`	The URL for the OAuth2 provider to give the authorization token.
`userinfoEndpoint`	`payara.security.openid.provider.userinfoEndpoint`	`true`	An OAuth 2.0 Protected Resource that returns Claims about the authenticated End-User.
`endSessionEndpoint`	`payara.security.openid.provider.endSessionEndpoint`	`false`	OIDC Provider's endpoint to notify that the End-User has logged out of the site and might want to log out of the OIDC Provider as well.
`jwksURI`	`payara.security.openid.provider.jwksURI`	`true`	An OIDC Provider's JSON Web Key Set document.
`scopesSupported`	`payara.security.openid.provider.scopesSupported`	recommended	List of the OAuth 2.0 scope values that this server supports, e.g. `openid`.
`responseTypesSupported`	`payara.security.openid.provider.responseTypeSupported`	`true`	List of the OAuth 2.0 response_type values that this OIDC Provider supports, e.g. `code`, `id_token`, `token id_token`.
`subjectTypesSupported`	`payara.security.openid.provider.subjectTypesSupported`	`true`	List of the Subject Identifier types that this OIDC Provider supports. Valid types include `pairwise` and `public`.

Option	MP Config property	Required	Description
`idTokenSigningAlgValuesSupported`	`payara.security.openid.provider.idTokenSigningAlgorithmsSupported`	`true`	List of the JWS signing algorithms (algorithm values) supported by the OIDC Provider for the ID Token to encode the Claims in a JWT, e.g. `RS256`.
`idTokenEncryptionAlgValuesSupported`	`payara.security.openid.provider.idTokenEncryptionAlgValuesSupported`	`false`	List of the JWE encryption algorithms (`alg` values) supported by the OIDC Provider for the ID Token to encode the Claims in a JWT.
`idTokenEncryptionEncValuesSupported`	`payara.security.openid.provider.idTokenEncryptionEncValuesSupported`	`false`	List of the JWE encryption algorithms (`enc` values) supported by the OIDC Provider for the ID Token to encode the Claims in a JWT.
`claimsSupported`	`payara.security.openid.provider.claimsSupported`	recommended	List of the Claim Names of the Claims that the OIDC Provider MAY be able to supply values for. Note that for privacy or other reasons, this might not be an exhaustive list.

Client Secret Aliasing

The client secret can be input directly, or for added security it can be aliased using "Microprofile Config"

> **NOTE** Environment Variables and System Properties can be used indirectly through MicroProfile Config.

Fetch Caller Data

As the OpenId Connect Client is built on top of Jakarta EE Security API, therefore `jakarta.security.enterprise.SecurityContext` interface can provide caller info which is available as a CDI bean and can be injected into any context-aware

instance.

The Payara Public API also provides a `fish.payara.security.openid.api.OpenIdContext` interface which is also available as a CDI bean and consist of the following methods:

- The `getCallerName()` method - Gets the caller name of the currently authenticated user.
- The `getCallerGroups()` method - Gets the groups associated with the caller.
- The `getSubject()` method - Subject Identifier. A locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the Client.
- The `getTokenType()` method - Gets the token type value. The value MUST be `Bearer` or another token type value that the client has negotiated with the authorization server.
- The `getAccessToken()` method - Gets the authorization token that was received from the OpenId Connect provider.
- The `getIdentityToken()` method - Gets the identity token that was received from the OpenId Connect provider.
- The `getRefreshToken()` method - Returns the refresh token that is used by OIDC client to get a new access token.
- The `getExpiresIn()` method - Return the time that the access token is granted for, if it is set to expire.
- The `getClaimsJson()` method - Gets the User Claims JSON that was received from the `userinfo` endpoint.
- The `getClaims()` method - Gets the User Claims that were received from the `userinfo` endpoint.
- The `getProviderMetadata()` method - The OpenId Connect Provider's metadata document fetched via provider URI.

User Information from the ID Token

The `userClaimsFromIDToken` attribute that belongs to the `@OpenIdAuthenticationDefinition` will instruct the container to retrieve the user information details from the ID Token instead of calling the `userinfo` endpoint, as defined by the OpenID Connect standard workflow.

As such, this behaviour is non-standard and should be used on special cases.

> **TIP** To properly connect an application with Microsoft ADFS this property is required because by default Microsoft ADFS doesn't allow calls to the `userinfo` endpoint.

Table 203. Configuration Option

Option	MP Config Property Name	Required	Description	Default value
`userClaimsFromIDToken`	`payara.security.openid.userClaimsFromIDToken`	`false`	Instructs the container to retrieve user information from the ID Token	`false`

Example

```
@OpenIdAuthenticationDefinition(
        providerURI = "https://sample-openid-server.com",
        clientId = "87068hgfg5675htfv6mrucov57bknst.apps.sample.com",
        clientSecret = "{my-secret}",
        redirectURI = "${baseURL}/callback",
        userClaimsFromIDToken=true
)
public class SecurityBean {

}
```

Disable Scope Validation

By default, the OpenID Connect connector validates that a scope is reported as a supported scope by the provider. However, some providers support more scopes than they actually report as supported scopes.

To disable the validation and allow using such scopes, it's possible to use the

`disableScopeValidation` property of `OpenIdProviderMetadata`. See the Healthcheck Service Configuration section for more details about this option.

Example

```
@OpenIdAuthenticationDefinition(
       providerURI = "https://sample-openid-server.com",
       clientId = "87068hgfg5675htfv6mrucov57bknst.apps.sample.com",
       clientSecret = "{my-secret}",
       redirectURI = "${baseURL}/callback",
       providerMetadata = @OpenIdProviderMetadata(disableScopeValidation = true))
public class SecurityBean {

}
```

Bearer Authentication and Authorization

In order to authenticate and authorize calls between services using the OpenID mechanism, it is possible to use authorization compatible with RFC 6750.

In this case, the access token presented to the resource service is an JWT token that is used to verify that the caller has access to OAuth2 protected resources.

Obtaining JWT Token

Obtaining the token is specific to the OAuth provider and the application. The usual approach is using Client Credentials Grant, where an application posts its clientId and secret to identity provider and receives access and refresh tokens in return.

Passing Token to the Resource Service

The obtained access token is passed with every request to the resource service by adding it into the `Authorization` HTTP header:

```
Authorization: Bearer access__token
```

Processing Bearer Authorization

When the `Bearer` authorization header is present in the request, the provided token is verified. It's validated that it comes from the expected issuer and hasn't expired.

NOTE	Compared to the normal browser flow, no groups are automatically assigned to the identity. The reason for this is that machine-to-machine

> communication tends to be much more fine-grained and services might want to check more claims, such as `audience`.

The resource service is required to map the information in the JWT token to groups utilizing the `IdentityStore` interface. The OpenID connector provides the following classes to make this process possible:

`AccessTokenCallerPrincipal`: Caller principal subclass that contains access to all claims of passed JWT token **`BearerGroupIdentityStore`**: A convenience base implementation of the necessary Jakarta EE security identity store.

```
@ApplicationScoped
@DeclareRoles({"user", "calendar-reader"})
public class Auth0BearerIdentityStore extends BearerGroupsIdentityStore {

    @Override
    protected Set<String> getCallerGroups(AccessTokenCallerPrincipal
callerPrincipal) {
        if (callerPrincipal.hasAudience("https://example.org/api/user")) {
            // if the token is for USER api, set this group
            return Set.of("user");
        }
        if (callerPrincipal.hasAudience("https://example.org/api/delegate")
                // delegate API is further constrained by scope
                && callerPrincipal.getAccessToken().getScope().contains
("read:calendar")) {
            return Set.of("calendar-reader");
        }
        return Set.of();
    }
}
```

TIP

> Azul Payara also provides similar functionality by way of the MicroProfile JWT Authentication specification, which is limited only to securing JAX-RS resources. On the other hand, the OpenID Connect Bearer Authentication and Authorization feature is better aligned with the OpenID Connect support in Azul Payara and can also be used to secure other web resources like Jakarta Servlets, for example.

Integration with Specific Providers

Google Integration

The Payara Public API provides built-in support for Google OpenID Provider via the `@GoogleAuthenticationDefinition` annotation.

Request Refresh Token

To enable the refresh token feature, set the `tokenAutoRefresh` to true and add the `access_type` parameter value to `offline` so that application can refresh access tokens when the user is not present at the browser.

If application requests `offline` access then the application can receive access and refresh token. Once the application has a refresh token, it can obtain a new access token at any time or as older ones expire.

Otherwise, if an application requests `online` access, your application will only receive an access token

```
@GoogleAuthenticationDefinition(
    providerURI="#{openidConfigBean.tokenEndpointURL}",
    clientId="#{openidConfigBean.clientId}",
    clientSecret="#{openidConfigBean.clientSecret}",
    tokenAutoRefresh = true,
    extraParameters = {"access_type=offline", "approval_prompt=force"}
)
public class SecurityBean {
}
```

Azure AD Integration

The Payara Public API also provides built-in support for Azure AD OpenID Provider via the `@AzureAuthenticationDefinition` annotation.

Request Refresh Token

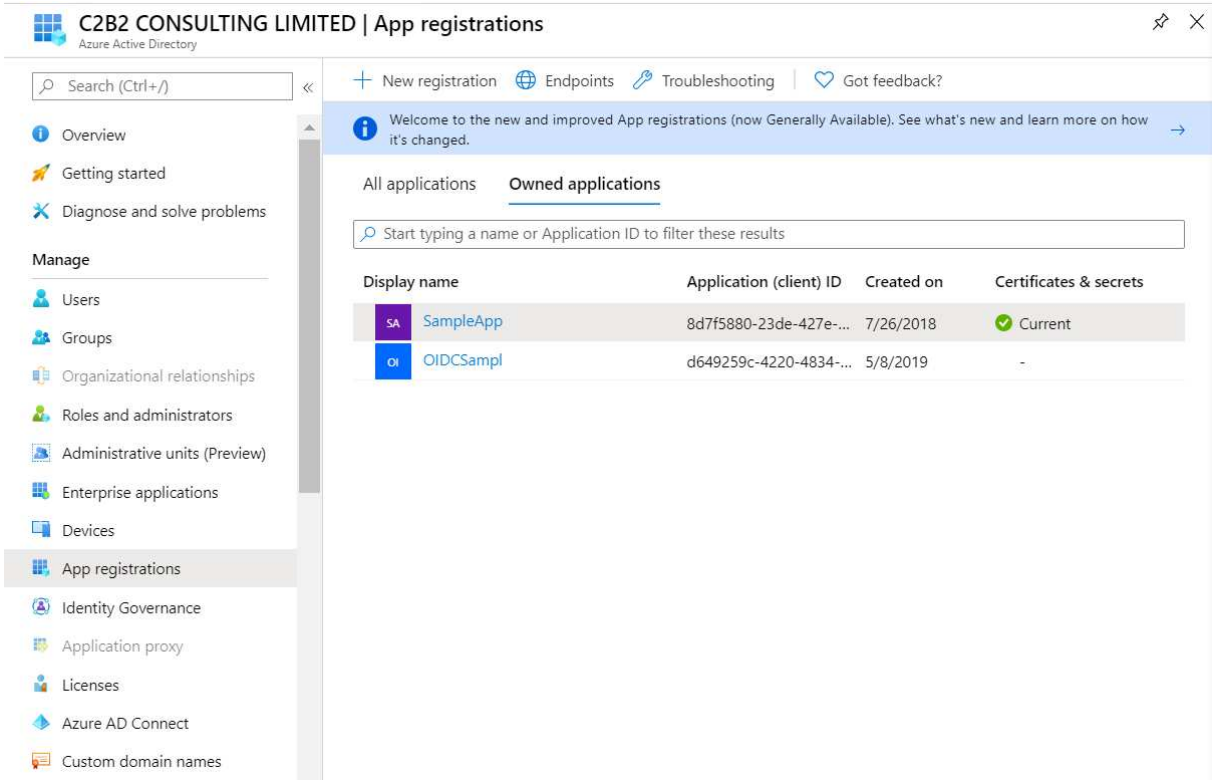
To receive the refresh token, set the `tokenAutoRefresh` to true and explicitly add the `offline_access` scope to the definition.

```
@AzureAuthenticationDefinition(
    providerURI="#{openidConfigBean.tokenEndpointURL}",
    clientId="#{openidConfigBean.clientId}",
    clientSecret="#{openidConfigBean.clientSecret}",
    tokenAutoRefresh = true,
    scope = {OPENID_SCOPE, EMAIL_SCOPE, PROFILE_SCOPE, OFFLINE_ACCESS_SCOPE}
)
```

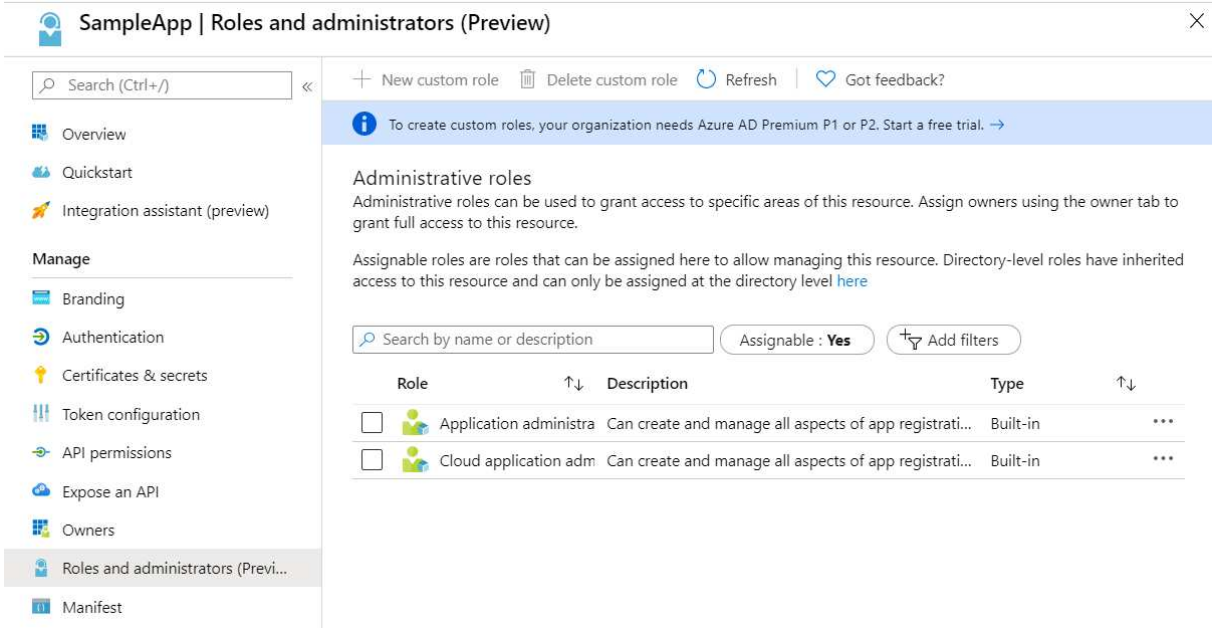
```
public class SecurityBean {
}
```

Groups Mapping

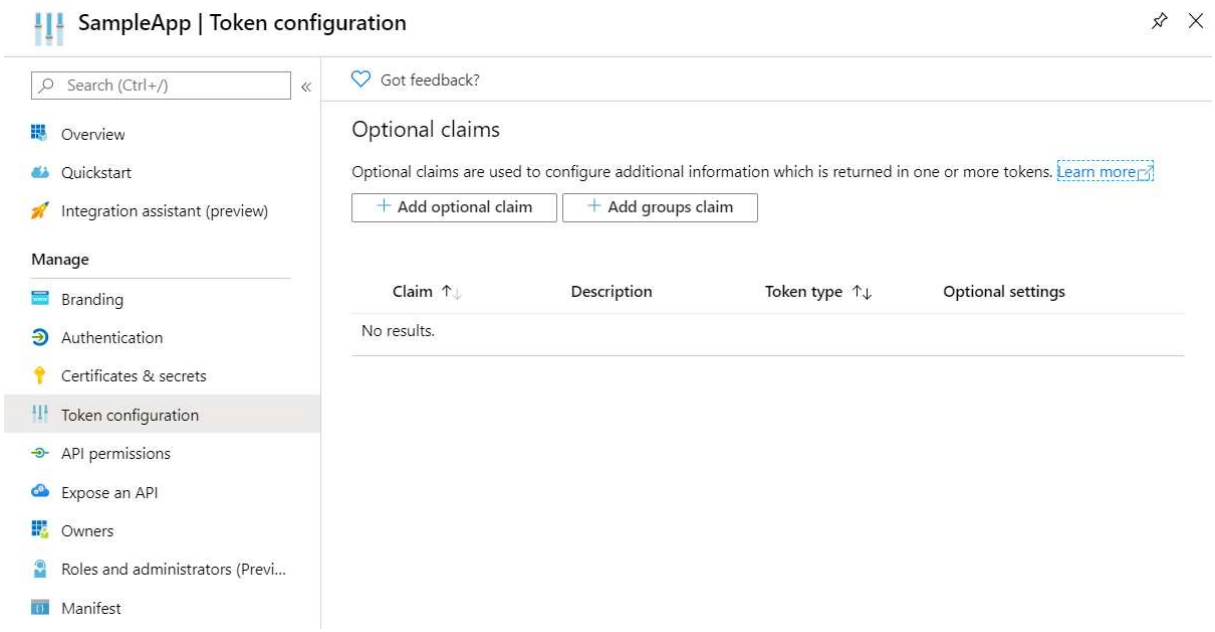
1. To add the groups to the registered application, Sign in to the Azure portal > Azure Active Directory Manage App registrations select your application:



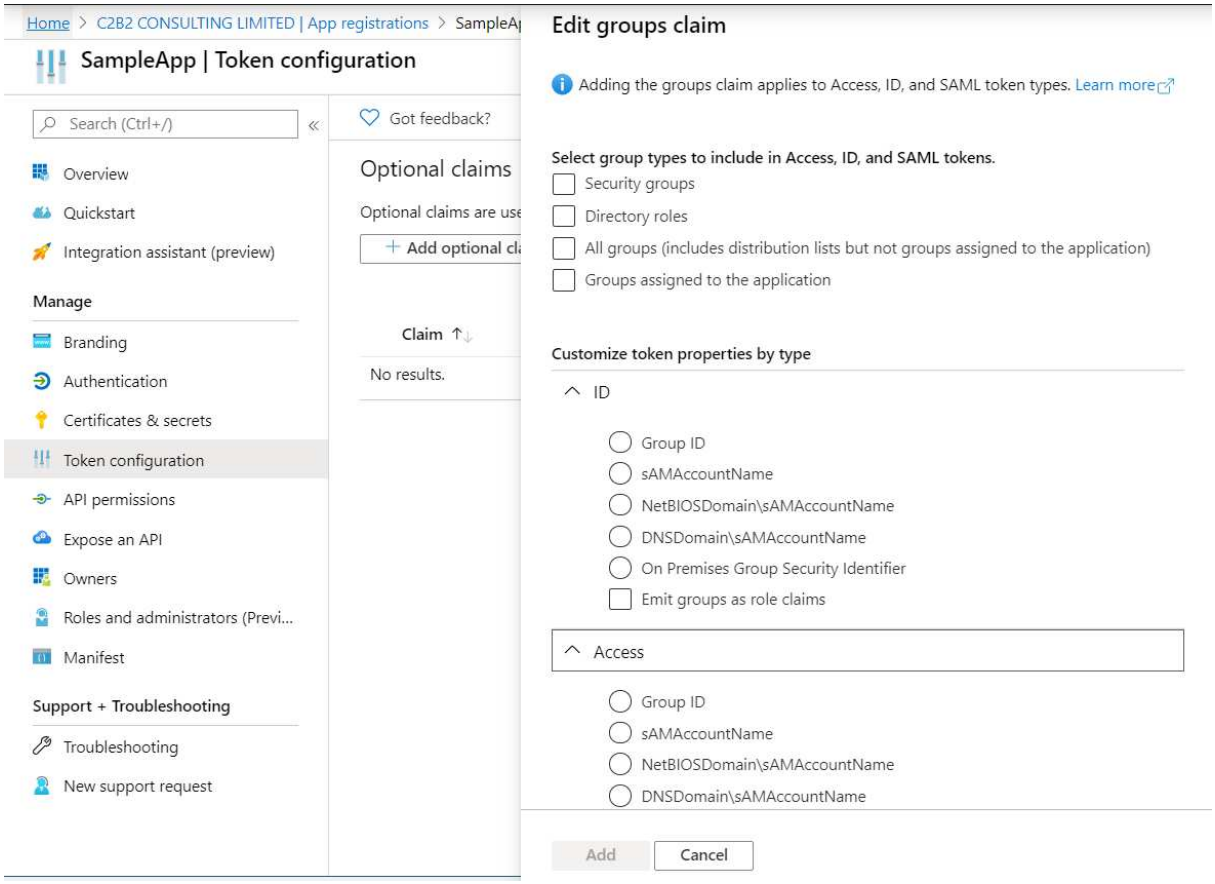
2. You may also add the custom roles via **Roles and administrators** under the **Manage** section:



3. Now to map group claims, select **Token configuration** under the **Manage** section:



4. Press **Add groups claim** button to select group types and customize the ID and/or Access Token properties:



5. Groups claim can also be defined via Azure **Manifest** under the **Manage** section which is a JSON configuration file.
6. To retrieve and map the caller name & groups from token claims, set the caller name & group claim definition to `preferred_username` & `groups`.

```
@AzureAuthenticationDefinition(
    providerURI="#{openidConfigBean.tokenEndpointURL}",
    clientId="#{openidConfigBean.clientId}",
    clientSecret="#{openidConfigBean.clientSecret}",
    claimsDefinition = @ClaimsDefinition(
            callerGroupsClaim = "groups",
            callerNameClaim = "preferred_username"
    )
)
public class SecurityBean {
}
```

Microsoft ADFS Integration

To enable integration for Microsoft ADFS it is needed to use the `userClaimsFromIDToken` annotation attribute.

See User Information from the ID Token for more information.

Azure AD Scope Validation

To disable the scope validation for Azure AD integration it is needed to use the `disableScopeValidation` annotation attribute.

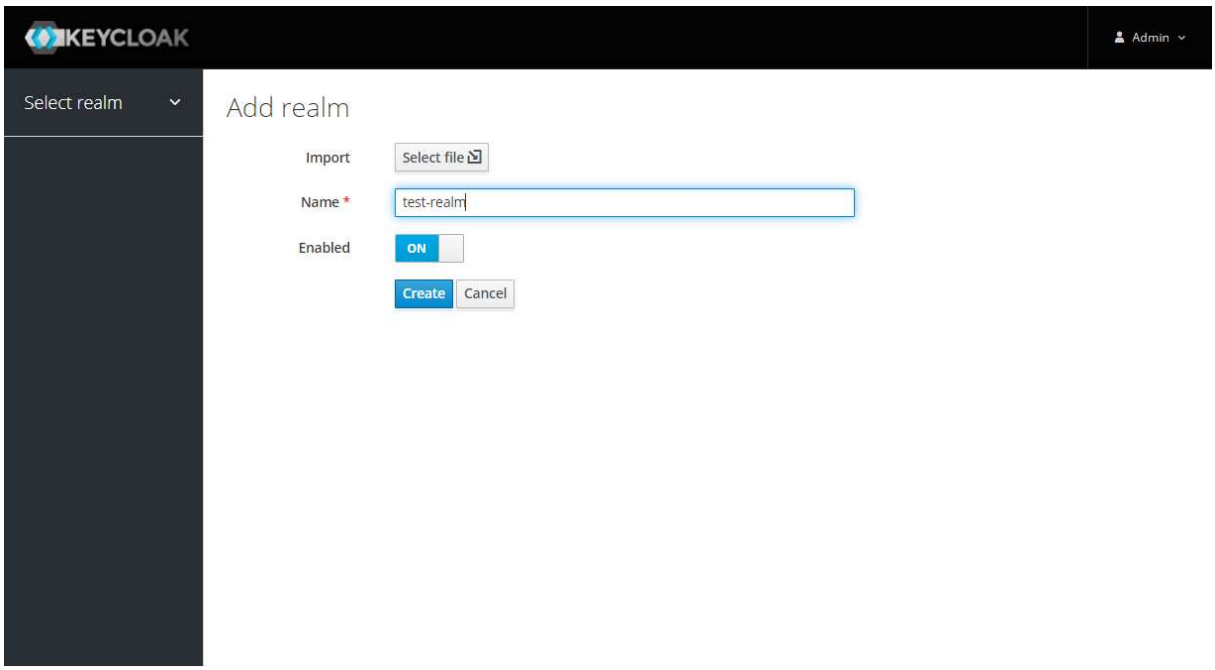
See Disable Scope Validation for more information.

Keycloak Integration

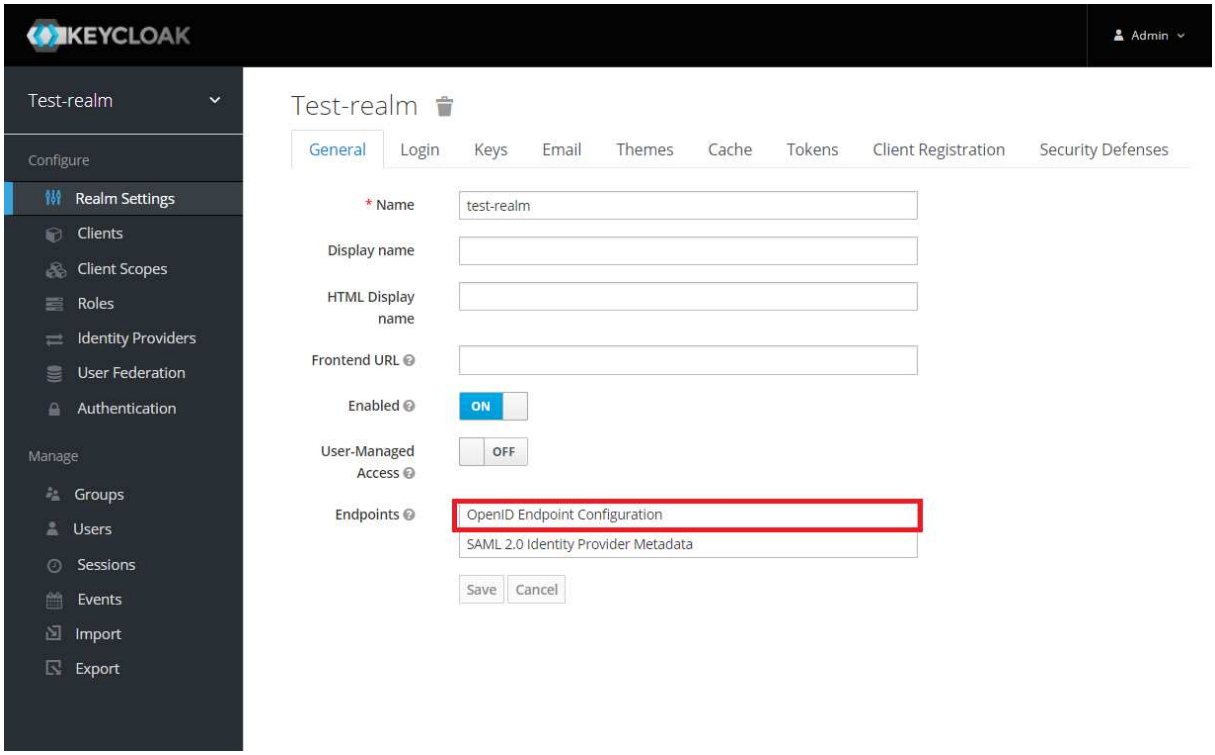
Keycloak is an Open Source Identity and Access Management Server, which is a OAuth2 and OpenID Connect(OIDC) protocol complaint. In this section,

the basic steps are described to set up a Keycloak OpenId provider:

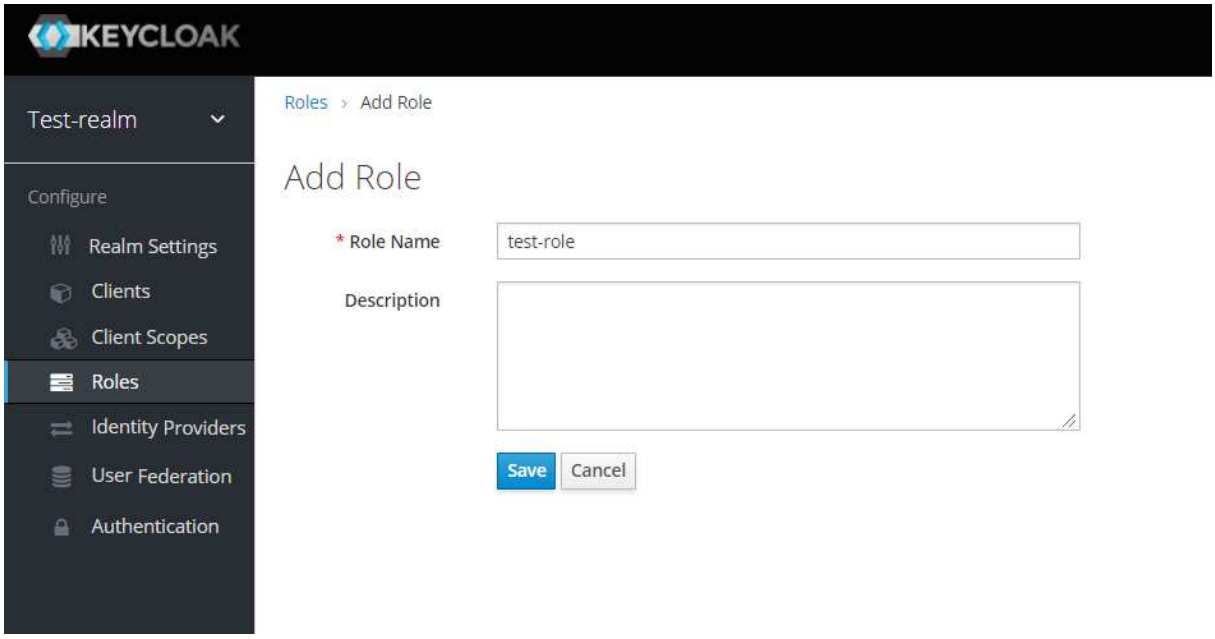
1. Refer to the Keycloak getting started documentation to run and setup Keycloak.
2. After Keycloak setup done, login to Keycloak admin console and add the new realm by pressing the **Add Realm** button:



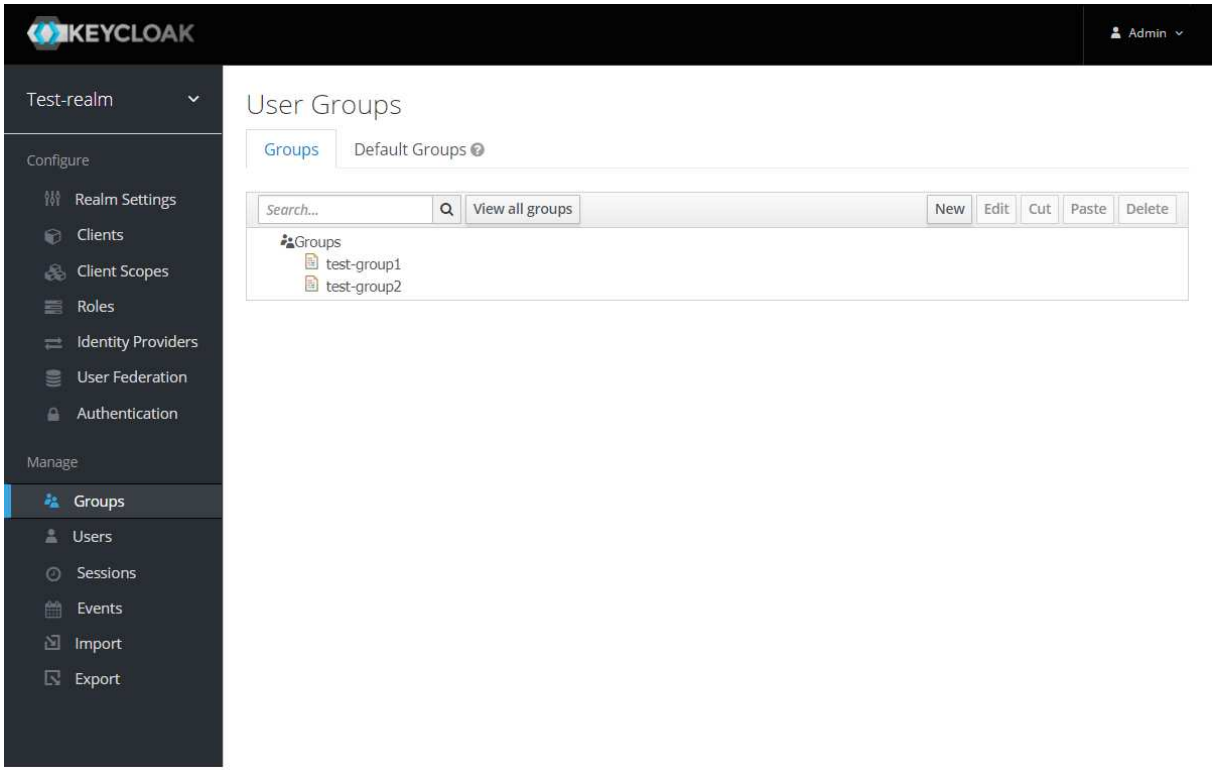
3. Copy the OpenId endpoint configuration URL from endpoint section:



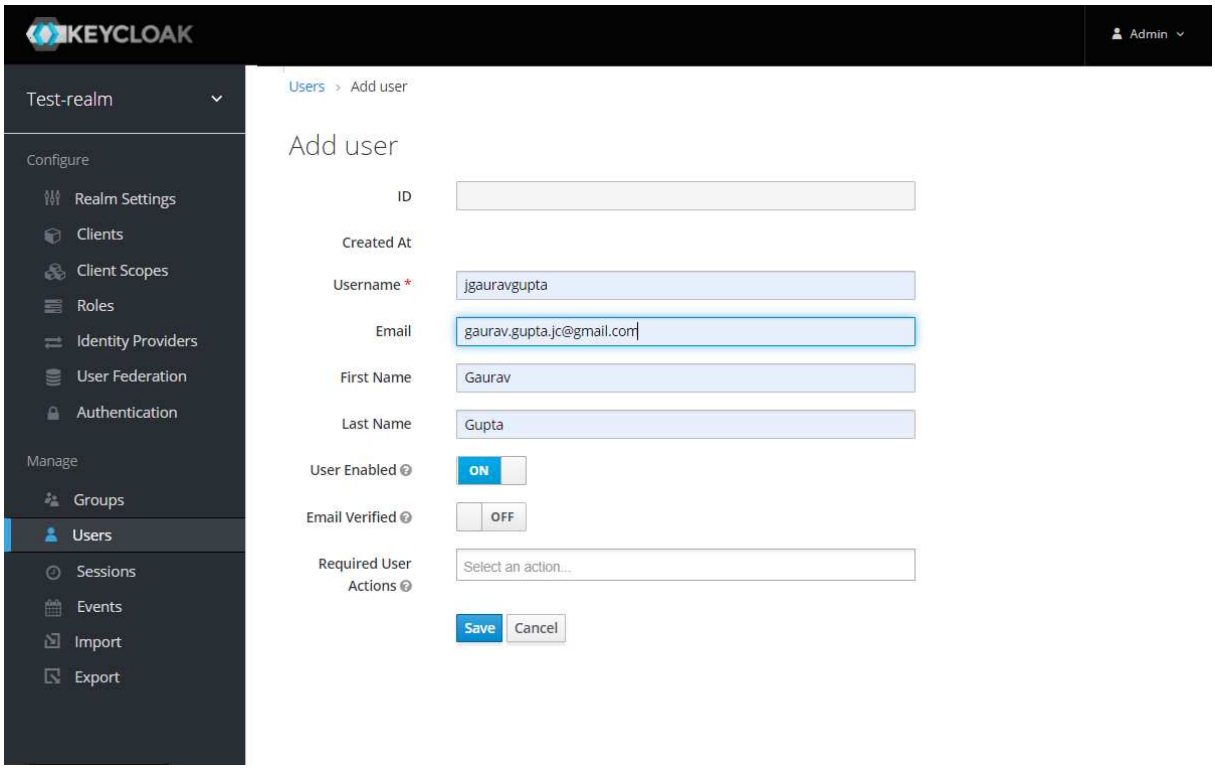
4. Now add the **Role** that will be used by the application to define which users will be authorized to access the application.



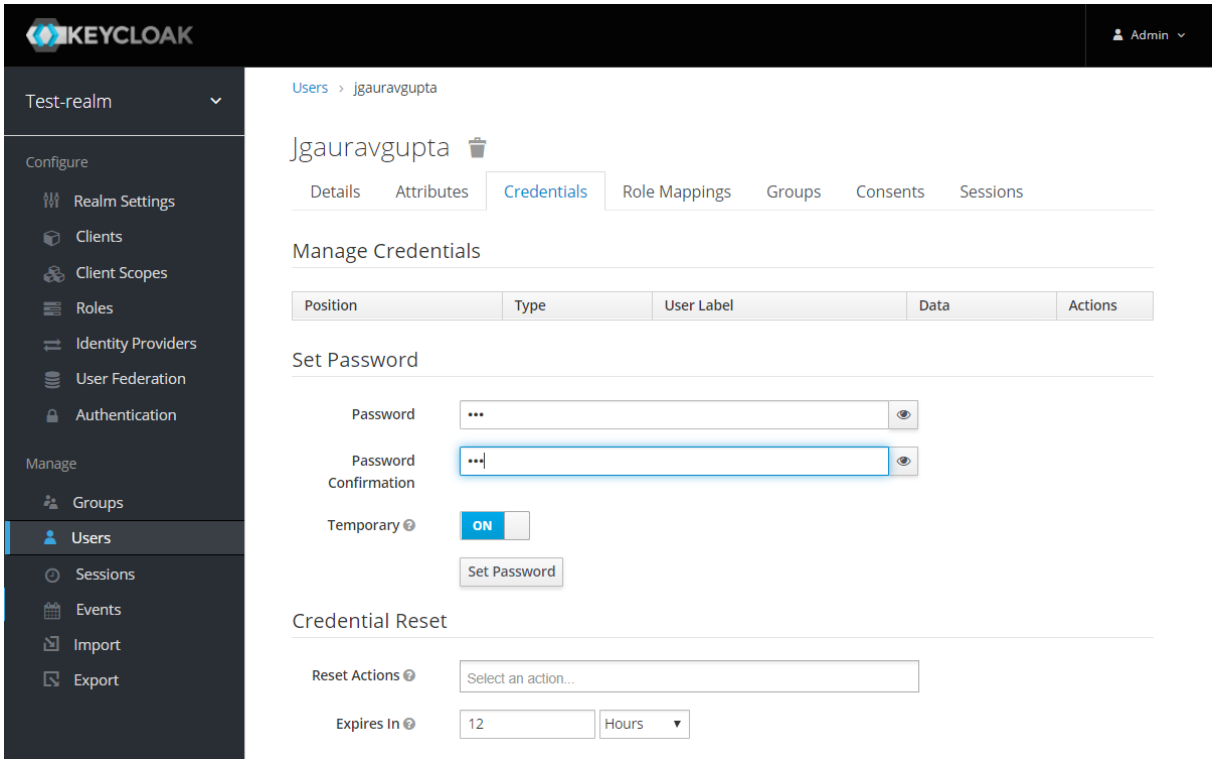
5. Create the **Groups**:



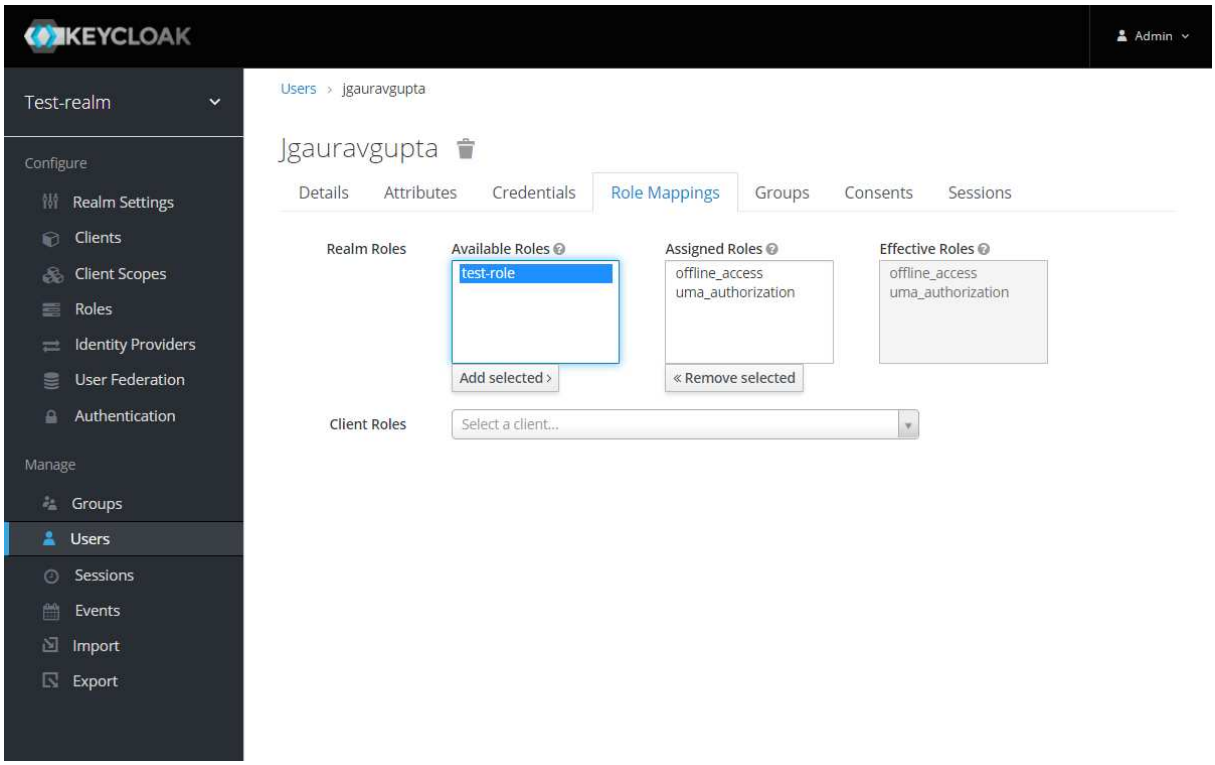
6. Add the **User**:



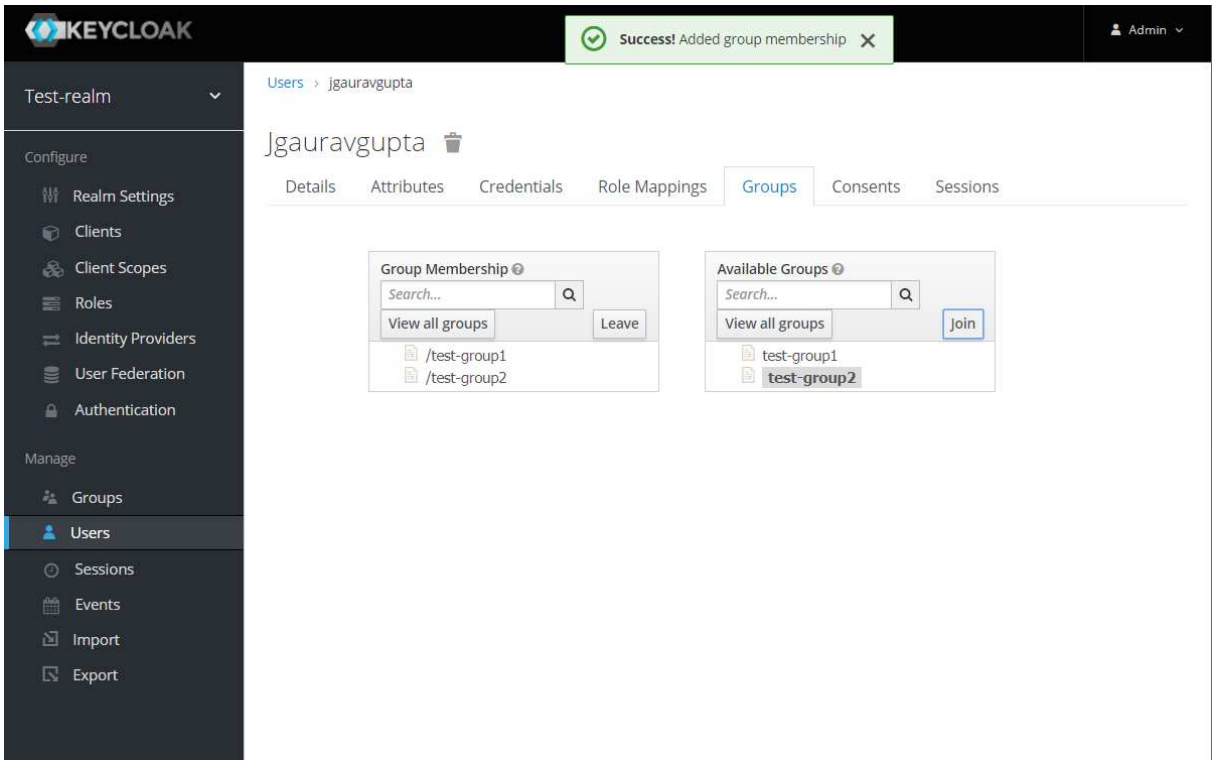
7. After the user is created, set a new **password** for the user:



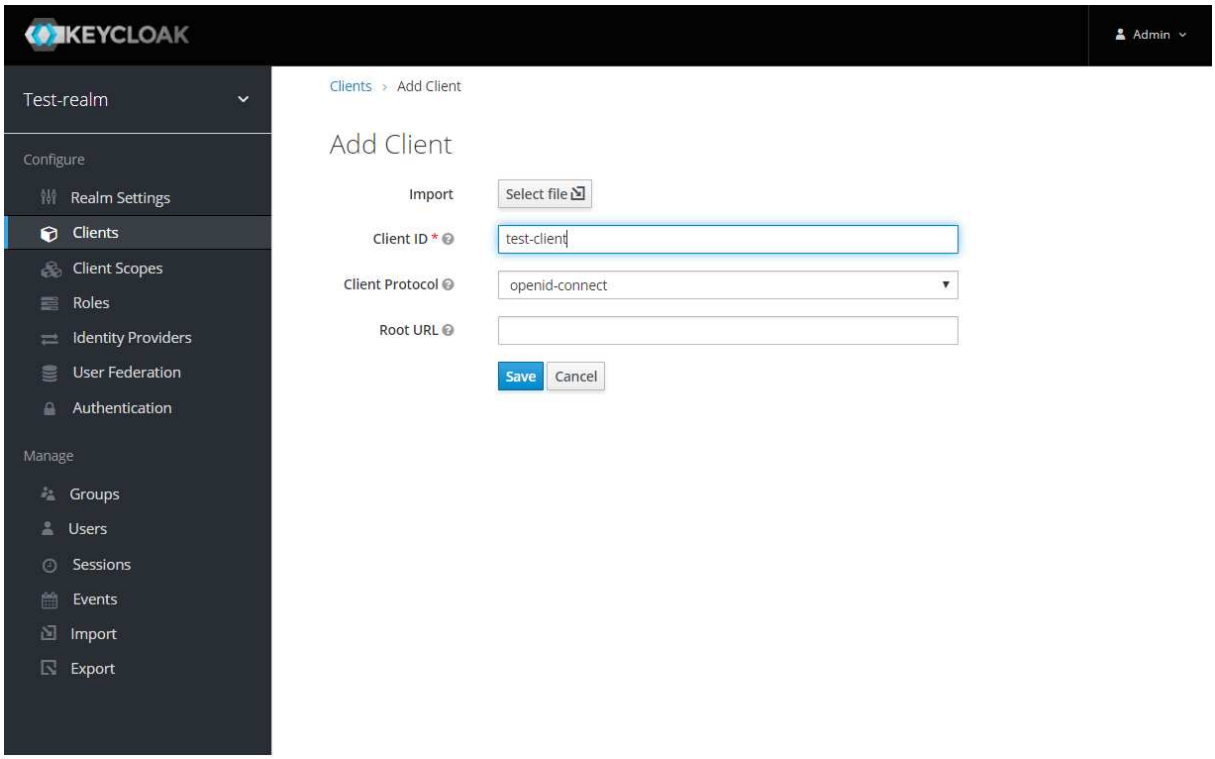
8. Now map the user to roles. Click on **Role Mappings** tab and assign the roles to the user from the available roles:



9. Assign the user to the groups. Click on **Groups** tab and join the groups from the available groups:

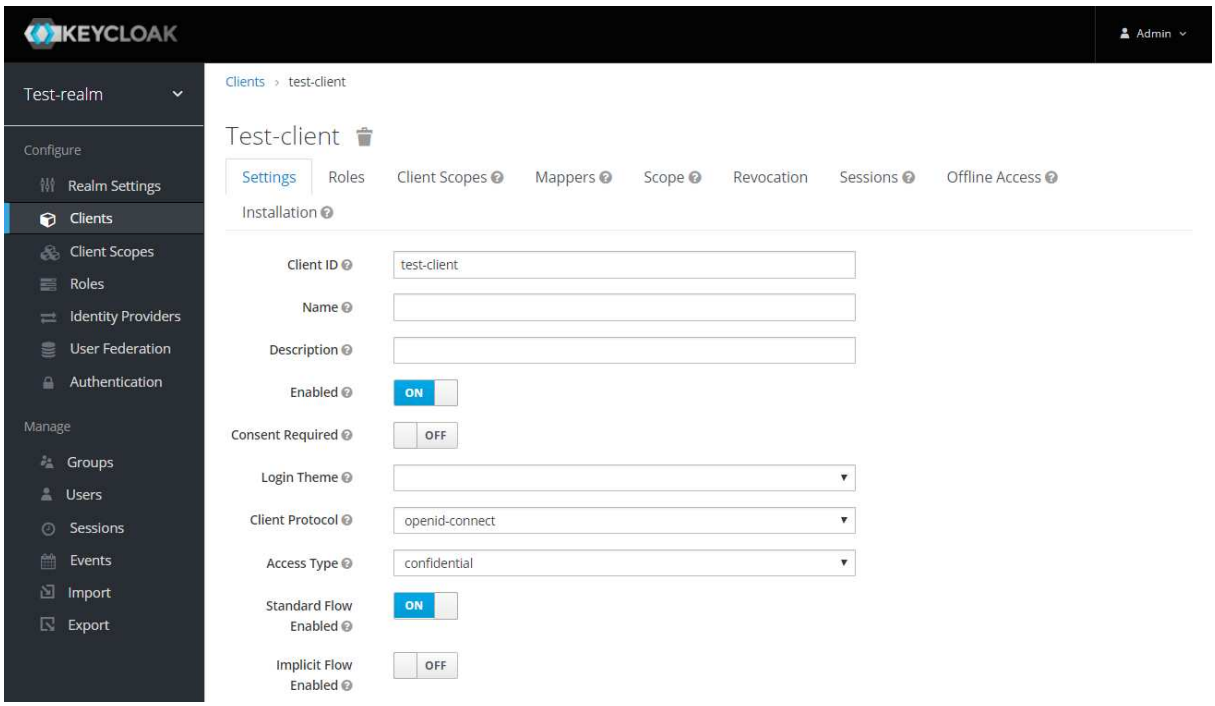


10. Create the OpenId Client by clicking the **Client** option from sidebar and press the **create** button:

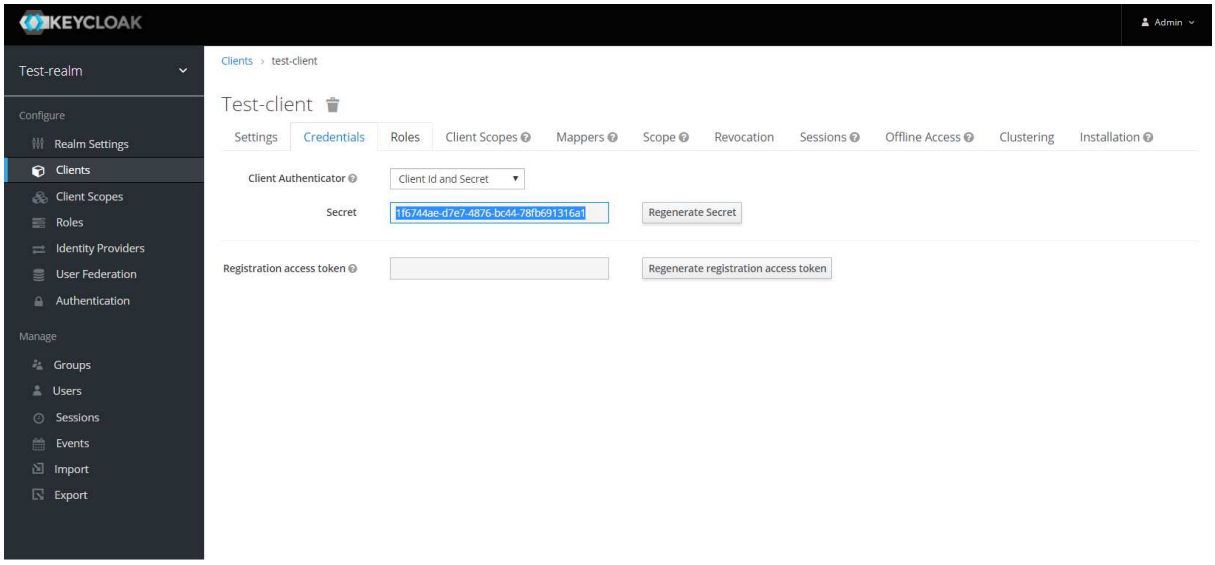


Enter the Client ID and select the Client Protocol **openid-connect** and press **Save**.

1. After the OpenId client is created change its **Access Type** to **confidential** and enter the valid **Redirect URIs**:



2. Next copy the client **secret** from **Credentials** tab.

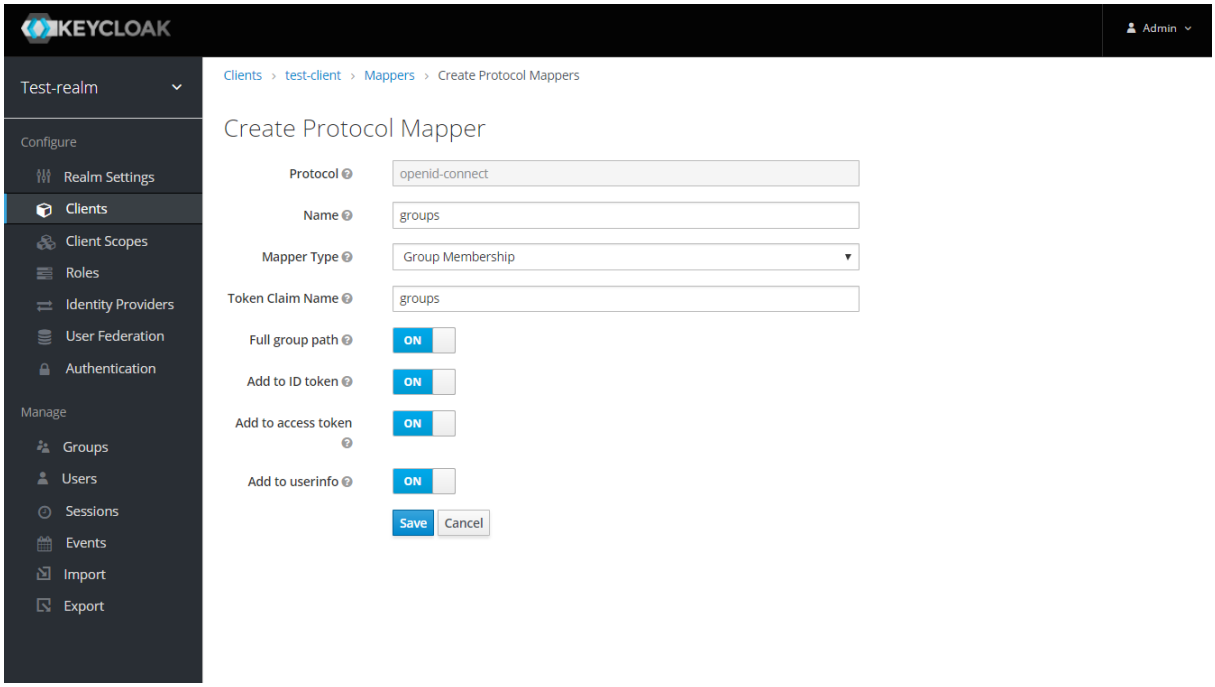


Here's an example that configures a OpenID Connect client for Keycloak provider. To test the KeyCloak OpenId provider, enter the copied client secret, client ID (client name) and the endpoint configuration URL:

```
@OpenIdAuthenticationDefinition(
    providerURI = "https://${keycloak-host}:${keycloak-port}/auth/realms/test-
realm",
    clientId = "test-client",
    clientSecret = "1f6744ae-d7e7-4876-bc44-78fb691316a1"
)
public class SecurityBean {
}
```

Groups Mapping

1. To get the groups details in token claims, navigate to KeyCloak admin console **OpenId Client** **Mappers** tab press **Create** button Select **Group Membership** mapper type enter the **Name** and **Token Claim Name** > press **Save**.



2. To retrieve and map the caller name & groups from token claims, set the caller name & group claim definition to `preferred_username` & `groups`.

```
@OpenIdAuthenticationDefinition(
    providerURI = "https://${keycloak-host}:${keycloak-port}/auth/realms/test-realm",
    clientId = "test-client",
    clientSecret = "1f6744ae-d7e7-4876-bc44-78fb691316a1",
    claimsDefinition = @ClaimsDefinition(
            callerGroupsClaim = "groups",
            callerNameClaim = "preferred_username"
    )
)
public class SecurityBean {
}
```

See Also

- OpenID Connect Core
- KeyCloak Documentation
- Jakarta Security API
- "MicroProfile JWT Authentication API"

Rolespermitted Support

The Payara Public API provides a `@RolesPermitted` annotation that creates an

authorisation interceptor for usage with CDI beans. This largely works in the same way as the `@RolesAllowed` common annotation, which in Java EE/Jakarta EE is primarily used by the EJB specification.

Usage

The authorisation interceptor is defined through the `@RolesPermitted` annotation. Specifying this on an interceptable method of a non-passivation capable CDI bean will make that method eligible for interception by the corresponding interceptor. This interceptor will check that the caller of a method is any or all of the specified roles. If this holds, the call to the method proceeds. If not, an exception of type `CallerAccessException` is thrown.

NOTE	The annotation can alternatively be placed at the class level of a CDI bean, in which case it will apply to all interceptable methods of that bean.

NOTE	Examples of a non-passivation capable CDI bean are beans defined in the Request and Application scope.

Example

Here's an example of defining the roles authorized to call a single method:

```
@RequestScoped
public class TestRolesPermitted {

    @Inject
    Principal principal;

    @RolesPermitted("payaraAdmin")
    public String getUserName() {
        return principal.getName();
    }
}
```

See this sample project for a more detailed example.

Configuration

The `@RolesPermitted` annotation has several configuration options.

They are detailed below.

Table 204. Configuration Options

Option	Type	Required	Description
`value`	`String`	true	A static list of comma-separated roles which are allowed to access this method, or an EL expression used to resolve this list at runtime (*see below*).
`semantics`	`String`	no	Whether accessing caller must be in any one of the given roles (`OR`) or all given roles (`AND`). Defaults to `OR`.

Dynamic Roles

The `@RolesPermitted` annotation also allows to dynamically specify authorization roles that are allowed to call bean methods at runtime. With this feature, is possible to instruct the container to allow specific roles based on specific contextual information.

Dynamic roles are defined using an *Expression Language (EL) 3.0* `String` value which should resolve to a valid `String` array or `Collection` that corresponds to the roles permitted to call the corresponding method. This expression value has contextual access to any CDI-scoped bean at the time of the method call.

NOTE As a reminder, you can use the `self` bean name in order to refer to the method's bean and access any of its methods and properties at runtime.

IMPORTANT In order to access the values of any relevant parameters via CDI beans in the method's signature, annotate them with the `jakarta.inject.Named` CDI annotation, and assign them the name which will be used in the expression value. If no names are provided for these parameters and the method only has one parameter, then it will be available as a bean with the `param` name.

Examples

Basic Example

The following example illustrates how to dynamically resolve a specific role based on the data received in the parameter. Assume that the application has declared multiple roles for each of one of the registered customers. Let's say that the application has 3 customers named `A`, `B` and `C`, in which case the following roles are defined:

- `Customer_A`
- `Customer_B`
- `Customer_C`

With each role granting authorization to each customer respectively. If we have a method for updating a customer's data based on a parameter received you can setup the corresponding authorization constraint like this:

```
@RolesPermitted("#{'Customer_'.concat(customer.name)}")
public void updateCustomer(@Named("customer") Customer customer){
    ...
}
```

As you can observe, the role name is evaluated at runtime by concatenating the `Customer_` prefix with the name of the customer, which is retrieved using the `customer` bean (which is injected in the context using the `@Named` annotation).

Advanced Expression Example

Based on the previous example, you can also let the corresponding expression be evaluated by calling a bean's method instead. Imagine that you could use a second variant, namely a product's name; in conjunction with the customer's name to evaluate authorization at runtime like this:

```
@RolesPermitted("#{self.findRole(customer, product)}")
public void updateCustomer(@Named("customer") Customer customer, @Named("product")
Product product){
    ...
}

public String findRole(Customer customer, Product product) {
    return customer.getType() + "_" + product.getModel();
```

```
}
```

By using the `self` bean name, we can call the bean's `findRole` method and pass it the two parameters defined in its signature.

Default Parameter Example

If no parameters are annotated with the `Named` annotation and the method has a single parameter then the `param` bean name can be used as a default reference like this:

```
@RolesPermitted("#{'CustomerCare_'.concat(param.name)}")
public void updateCustomer(Customer customer){
    ...
}
```

See Also

- @RolesAllowed annotation documentation

Jakarta EE Security Extensions

This section documents available extensions for custom authentication mechanisms and identity store configurations that can be used in any Jakarta EE application.

Custom Authentication Mechanisms

The Azul Payara Public API offers the following custom authentication mechanisms for use cases that aren't covered in the Jakarta EE Security API.

Client Certificate Authentication Mechanism

The Jakarta EE Security API doesn't include any specific authentication mechanisms for support client certificate validation, so the `@CertificateAuthenticationMechanismDefinition` can be used to instruct any application to initiate client certificate validation when any HTTP request is to be authenticated.

TIP	This authentication mechanism will mostly be used in conjunction with the `@CertificateIdentityStoreDefinition` (see below for more information), since certificates are probably stored in a client certificate

> security realm.

Configuration

No specific configuration is needed for this annotation.

Example

The following code sample illustrates how to configure a client certificate authentication mechanism:

```
@ApplicationScoped
@ApplicationPath("/rest")
@DeclareRoles({ "a", "b" })
@CertificateAuthenticationMechanismDefinition
@CertificateIdentityStoreDefinition("certificate-realm")
public class MyRestApp extends Application {
}
```

Custom Identity Stores

By default, the Jakarta EE Security API only instructs containers to implement identity stores for the following types:

- LDAP directories through the `@LdapIdentityStoreDefinition` annotation.
- Relational databases through the `@DatabaseIdentityStoreDefinition` annotation.

The following annotations serve to provide additional identity store configurations, by allowing the mapping an existing Azul Payara security realm

Basic Realm Identity Store

The `@RealmIdentityStoreDefinition` annotation allows users to plug-in any existing security realm as a valid identity store definition.

> **NOTE** No additional configuration settings or properties are available, so in case of needing further control over the store definition, see the more specific annotations defined below.

Configuration

The `@RealmIdentityStoreDefinition` annotation is configured with the options as shown below.

Option	Description	Default	Required
`value`	The name of an existing security realm.	<Default Realm>	`true`

Example

The following code sample illustrates how to configure the default `file` security realm as an identity store:

```
@ApplicationScoped
@ApplicationPath("/rest")
@DeclareRoles({ "a", "b"})
@BasicAuthenticationMechanismDefinition(realmName = "file")
@RealmIdentityStoreDefinition("file")
public class MyRestApp extends Application {
}
```

TIP

The `RealmIdentityStoreDefinition` is a **repeatable** annotation, hence multiple security realms can be used in sequence to define more than one identity store

```
@ApplicationScoped
@ApplicationPath("/rest")
@DeclareRoles({ "a", "b"})
@BasicAuthenticationMechanismDefinition
@RealmIdentityStoreDefinition("realm1")
@RealmIdentityStoreDefinition("realm2")
public class MyRestApp extends Application {
}
```

File Identity Store

The `@FileIdentityStoreDefinition` annotation allows users to plug in an existing file security realm (`com.sun.enterprise.security.auth.realm.file.FileRealm`) as a valid identity store definition.

IMPORTANT	If no realm is found with the defined name then a new realm will register on the server's configuration using the `create-auth-realm` asadmin command.

Configuration

The file realm identity store can be configured via both `@FileIdentityStoreDefinition` annotation attributes and MicroProfile Configuration properties. Here's a list of all available and equivalent settings:

Option	MP Config Property	Description	Default	Required
`value`		The name of the realm.	`file`	`true`
`file`	`payara.security.file`	The location of the file to store user credentials locally. If no file name is defined then the realm name is used as the file name.	<Realm Name>	`false`
`assignGroups`	`payara.security.file.assignGroups`	Users will get assigned membership to these groups automatically on successful authentication		`false`
`jaasContext`	`payara.security.file.jaasContext`	The JAAS Context of the file realm.	`fileRealm`	`false`

CAUTION	If both an annotation attribute and a MicroProfile Configuration property are defined for the same option then the configuration property always takes precedence.

Example

The following code sample illustrates how to configure a file realm identity store:

```
@ApplicationScoped
@ApplicationPath("/rest")
@DeclareRoles({ "a", "b"})
```

```
@BasicAuthenticationMechanismDefinition(realmName = "file-realm")
@FileIdentityStoreDefinition("file-realm")
public class MyRestApp extends Application {
}
```

Client Certificate Identity Store

The `@CertificateIdentityStoreDefinition` annotation allows users to plug-in an existing client `certificate` security realm (`com.sun.enterprise.security.auth.realm.certificate.CertificateRealm`) as a valid identity store definition.

IMPORTANT	If no realm is found with the defined name then a new realm will register on the server's configuration using the `create-auth-realm` asadmin command.

Configuration

The certificate realm identity store can be configured via both `@CertificateIdentityStoreDefinition` annotation attributes and MicroProfile Configuration properties. Here's a list of all available and equivalent settings:

Option	MP Config property	Description	Default	Required
`value`		The name of the certificate realm.	`certificate`	`true`
`assignGroups`	`payara.security.certificate.assignGroups`	Users will get assigned membership to these groups automatically on successful authentication.		`false`

CAUTION	If both an annotation attribute and a MicroProfile Configuration property are defined for the same option then the configuration property always takes precedence.

Example

The following code sample illustrates how to configure a certificate realm identity store:

```
@ApplicationScoped
@ApplicationPath("/rest")
@DeclareRoles({ "a", "b" })
@CertificateAuthenticationMechanismDefinition
@CertificateIdentityStoreDefinition("certificate-realm")
public class MyRestApp extends Application {
}
```

PAM Identity Store

The `@PamIdentityStoreDefinition` annotation allows users to plug-in an existing PAM (Privileged Access Management) security realm (`com.sun.enterprise.security.auth.realm.pam.PamRealm`) as a valid identity store definition.

IMPORTANT	If no realm is found with the defined name then a new realm will register on the server's configuration using the `create-auth-realm` asadmin command.

Configuration

The pam realm identity store can be configured via both `@PamIdentityStoreDefinition` annotation attributes and MicroProfile Configuration properties. Here's a list of all available and equivalent settings:

Option	MP Config property	Description	Default	Required
`value`		The name of PAM realm.		`true`
`assignGroups`	`payara.security.pam.assignGroups`	Users will get assigned membership to these groups automatically on successful authentication.		`false`
`jaasContext`	`payara.security.pam.jaasContext`	The JAAS Context of the PAM realm.	`pamRealm`	`false`

Example

The following code sample illustrates how to configure a PAM realm identity store:

```
@ApplicationScoped
@ApplicationPath("/rest")
@DeclareRoles({ "a", "b"})
@BasicAuthenticationMechanismDefinition(realmName = "pam-realm")
@PamIdentityStoreDefinition("pam-realm")
public class MyRestApp extends Application {
}
```

CAUTION	If both an annotation attribute and a MicroProfile Configuration property are defined for the same option then the configuration property always takes precedence.

Solaris Identity Store

The `@SolarisIdentityStoreDefinition` annotation allows users to plug in an existing Solaris security realm (`com.sun.enterprise.security.auth.realm.solaris.SolarisRealm`) as a valid identity store definition.

IMPORTANT	If no realm is found with the defined name then a new realm will register on the server's configuration using the `create-auth-realm` asadmin command.

Configuration

The Solaris realm identity store can be configured via both `@SolarisIdentityStoreDefinition` annotation attributes and MicroProfile Configuration properties. Here's a list of all available and equivalent settings:

Option	MP Config property	Description	Default	Required
`value`		The name of the Solaris realm.		`true`
`assignGroups`	`payara.security.solaris.assignGroups`	Users will get assigned membership to these groups automatically on successful authentication.		`false`

Option	MP Config property	Description	Default	Required
`jaasContext`	`payara.security.solaris.jaasContext`	The JAAS Context of the Solaris realm.	`solarisRealm`	`false`

> **CAUTION** If both an annotation attribute and a MicroProfile Configuration property are defined for the same option then the configuration property always takes precedence.

Example

The following code sample illustrates how to configure a Solaris realm identity store:

```
@ApplicationScoped
@ApplicationPath("/rest")
@DeclareRoles({ "a", "b"})
@BasicAuthenticationMechanismDefinition(realmName = "solaris-realm")
@SolarisIdentityStoreDefinition("solaris-realm")
public class MyRestApp extends Application {
}
```

MicroProfile

Eclipse Microprofile

What Is Eclipse Microprofile?

The MicroProfile project was launched in June 2016 as a collaborative effort between Java application server vendors and the enterprise Java community to enable fast innovation.

MicroProfile is a specification which provides a set of APIs focused on microservices. The complete MicroProfile API is provided as a Bill Of Materials(BOM) which can be used by developers as a generic dependency provided by a vendor of their choice.

Payara has been part of the MicroProfile project from its inception and, therefore, supports all specifications of the MicroProfile Umbrella project.

Payara Support for Microprofile

Version 6.1 Version 6.1 of MicroProfile is a minor release that updates Metrics, Config and Telemetry. Also, this version aligns with Jakarta EE 10 Core Profile and continues with the `jakarta.*` namespace alignment and allows to be certified with Java SE 11 or higher.

This is supported by Azul Payara Community starting from releases **7.2025.1** onwards

What Does Each Specification Contain?

Version	Specifications	Maven

6.1	• Jakarta EE Core Profile 10.0
	• Config 3.1
	• Metrics 5.1
	• Health Check 4.0
	• Fault Tolerance 4.0
	• JWT Authentication 2.1
	• OpenAPI 3.1
	• Rest Client 3.0
	• Telemetry Tracing 1.1
		`<dependency>`
		`  <groupId>org.eclipse.microprofile</groupId>`
		`  <artifactId>microprofile</artifactId>`
		`  <version>6.1</version>`
		`  <type>pom</type>`
		`  <scope>provided</scope>`
		`</dependency>`

Config

Eclipse Microprofile Config API

Azul Payara 7.2.0 provides MicroProfile Configuration 3.1

Background

The Config API was created to allow for the modification of application configuration from outside the application without needing the application to be repackaged.

The complete specification can be found on the Eclipse MicroProfile website.

Breaking changes introduced in MicroProfile Config 3.1 are listed in the official specification under Incompatible Changes

Config Sources

A `ConfigSource` is a definition of a source of configuration values that can be used by an application. In addition to the default ConfigSources outlined by the specification, the following are also supported ConfigSources:

> **NOTE** Where multiple configuration properties exist, the property from a source with a higher ordinal number is preferred.

Source	Ordinal	Description
`directory`	90	Reads configuration properties from files in a specified directory. Each file name is the configuration property name and the contents of the file are the config property value. This config source can be used to read Kubernetes secrets files. + You can find out more about directory config sources here: "Directory Config Sources"
`toml`	95	Reads configuration properties from a specified TOML configuration file. + You can find out more about TOML config sources here: "TOML Config Sources"
`password`	105	Retrieves and stores config values in the password alias store. These will be available to all server instances in a domain.
`domain`	110	Retrieves and stores config values in the domain.xml as key values. These will then apply to all instances within a domain
`jndi`	115	Retrieves and stores config values within the JNDI tree. The property name being the dotted name path in the JNDI tree and the value being stored in the JNDI tree. These are modifiable by current admin console functionality.
`config`	120	Retrieves and stores config values at the level of the named configuration in the domain.xml; `server-config`, for example. MicroProfile Configuration property values will apply to all instances that use the Payara configuration specified.

Source	Ordinal	Description
`server`	130	Retrieves and stores config values at the level of the server in the domain.xml. Property values will only apply on the named server instance
`application`	140	Retrieves and stores config values at the level of the application. Property values will only be available for the specified application on all server instances it is deployed
`module`	150	Retrieves and stores config values at the level of an individual module within an application. Property values will only be available for the specified module on all server instances it is deployed
`cluster`	160	Retrieves and stores config values in the **Hazelcast** cluster as name value pairs. These will be available for all server instances that are members of the cluster.
`payara-expression`	170	Reads configuration properties from a `payara-expression-config.properties` file. This should be stored in the same location as your `microprofile-config.properties` file This file can be used to reference properties containing variables which will be translated to their actual values upon retrieval. + The property values should be in the format referenced here: "Types of Variables"
`cloud`	180	Reads configuration properties from various Cloud provider secret stores. + You can find out more about our supported cloud config sources here: "Cloud Config Sources"
`jdbc`	190	Reads configuration properties from a specified database table. + You can find out more about JDBC config sources here: "JDBC Config Sources"

Source	Ordinal	Description
`ldap`	200	Reads configuration properties from a specified LDAP directory. + You can find out more about LDAP config sources here: "LDAP Config Sources"

Converters

In addition to the standard converters specified by the API, Azul Payara's implementation includes out-of-the-box converters for URL and `InetAddress`.

These have `@Priority` of `1`, the same as the default converters.

Azul Payara Configuration Properties

In addition to the standard configuration properties, Payara Server and Micro provide a number of out-of-the-box config properties you can use in your application to find information about the runtime environment.

Property	Description
`payara.instance.type`	The type of the runtime from `DAS`, `INSTANCE`, `EMBEDDED`, `MICRO`
`payara.instance.name`	The name of the instance e.g. `server`
`payara.instance.root`	The directory which is the root of the instance installation
`payara.domain.name`	The name of the Payara domain
`payara.domain.installroot`	The directory which is the root of the Payara software installation
`payara.config.dir`	The directory in which the server's configuration files reside
`payara.instance.starttime`	The start time of the server in `System.currentTimeMillis`
`payara.instance.config.name`	The name of the configuration this server is using
`payara.instance.admin.port`	The administration port number of this instance
`payara.instance.admin.host`	The host IP address which the admin listener is listening on
`payara.instance.http.port`	The HTTP port for http-listener

Property	Description
`payara.instance.http.address`	The IP address the http listener is listening on
`payara.instance.http.enabled`	Whether the http listener is enabled
`payara.instance.https.port`	The HTTP port for https-listener
`payara.instance.https.address`	The IP address the https listener is listening on
`payara.instance.https.enabled`	Whether the https listener is enabled

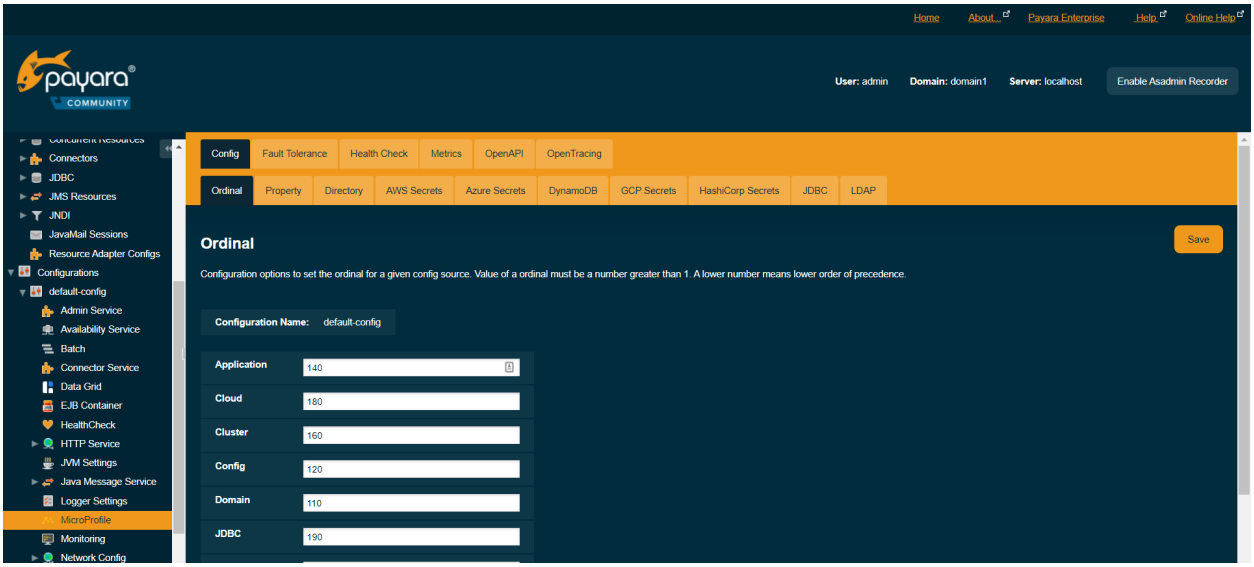
Payara Server offers additional facilities to configure specific settings of the MicroProfile Configuration API, described next.

Config Ordinal Configuration

Configuration variables ordinal values can be configured by using Admin Console or via the Asadmin CLI.

Using the Admin Console

To configure the ordinal value in the Admin Console, go to Configuration [instance-configuration (like server-config)] MicroProfile Config Ordinal:



Using Asadmin Commands

IMPORTANT	A `--source` value of `jndi` is not supported for these asadmin commands yet. The JNDI `ConfigSource` does exist and is

> usable from the Config API, though.

The following Asadmin commands can be used to manipulate configuration sources ordinal values:

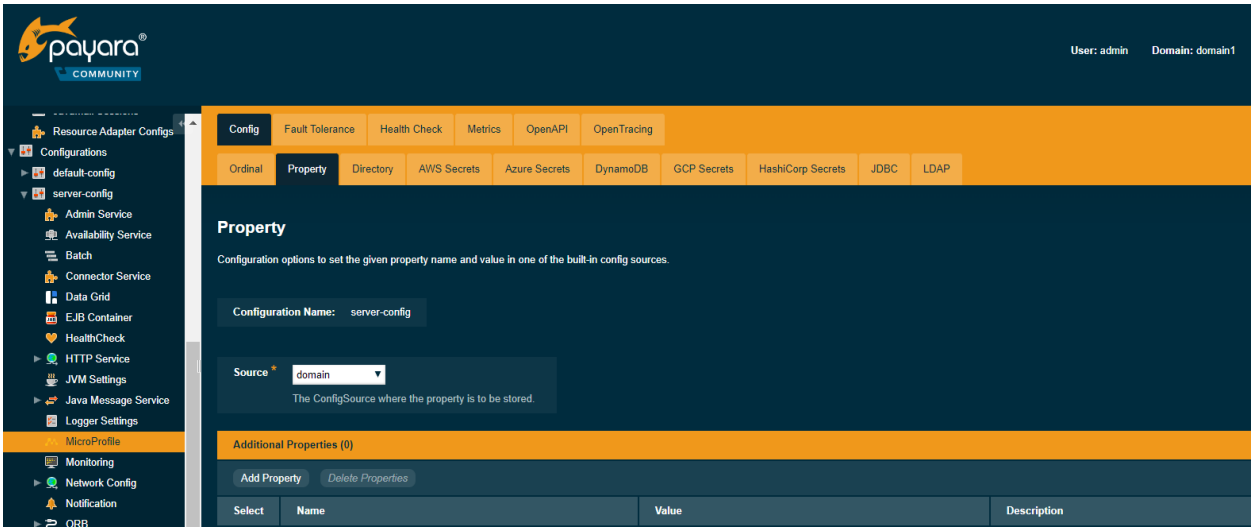
- set-config-ordinal
- get-config-ordinal

Configuration Property Manipulation

Configuration properties can be configured by using Admin Console or Asadmin commands.

Using the Admin Console

To configure the property in the Admin Console, go to Configuration [instance-configuration (like server-config)] MicroProfile Config Property:



Using Asadmin Commands

The following Asadmin commands can be used to directly configure MicroProfile configuration properties:

- set-config-property
- get-config-property

- delete-config-property

Cache Duration Configuration

The cache duration of the values of a configuration property can be configured by using Asadmin commands. By default, the cached value of properties have a TTL (time to live) of `60` seconds.

That means each individual property does not change for `60` seconds since it has been resolved before. Therefore, it can take up to `60` seconds for changes made visible by a source to become effective.

TIP	If properties were not resolved recently, the change can become visible faster than the cache duration or even immediately because some time already passed since they had been last resolved and cached.

Use the set-config-cache to modify this setting.

Cloud

Cloud Configuration Sources

The **Cloud Configuration Sources** can be used to extract configuration properties from a secret store or a database of several different cloud providers.

The following sources are currently available:

- "AWS Secrets Manager"
- "Azure Secrets"
- "DynamoDB Database"
- "GCP Secret Manager"
- "Hashicorp Secrets"

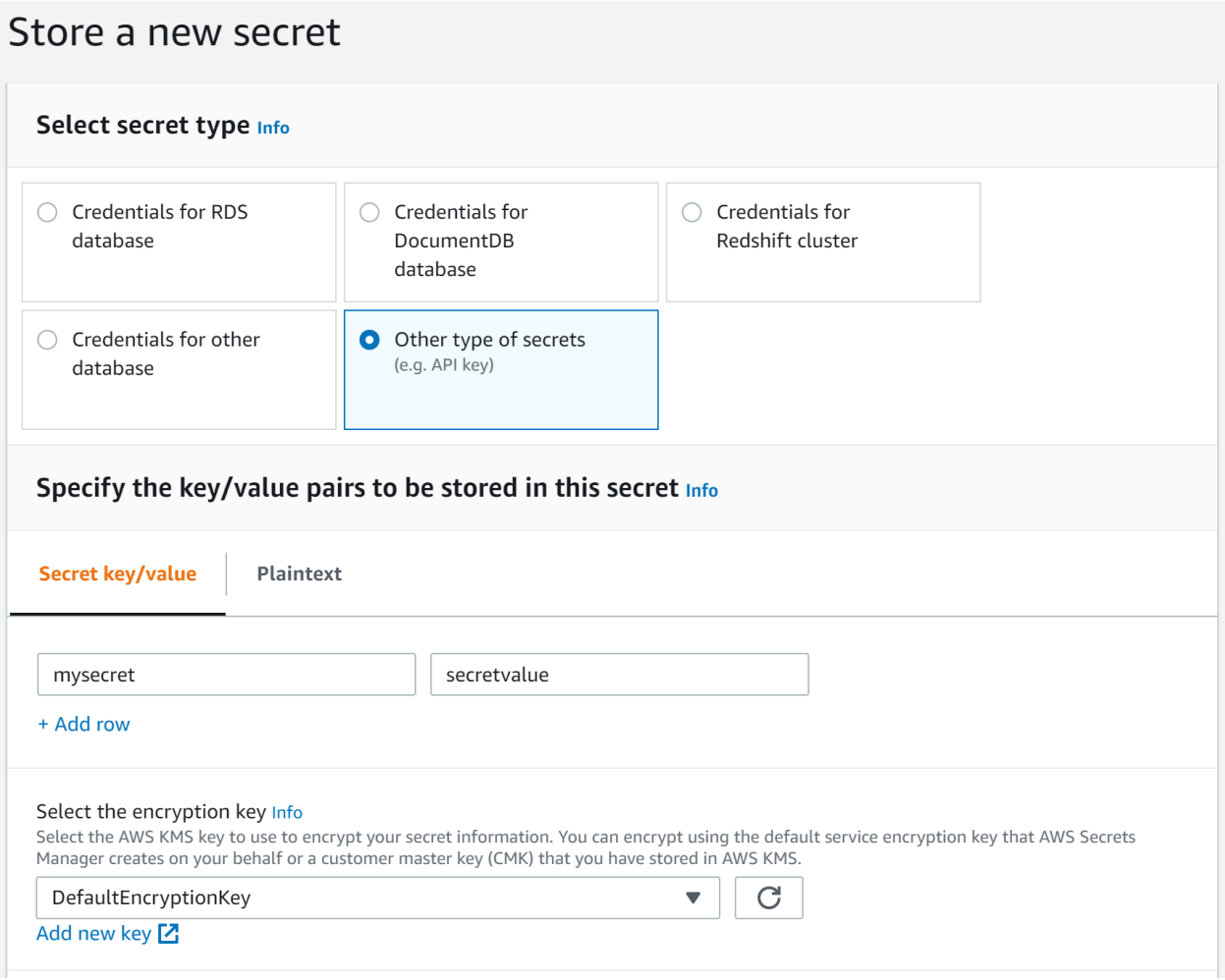
AWS Cloud Config Source

The AWS cloud config source takes configuration properties from AWS Secrets Manager.

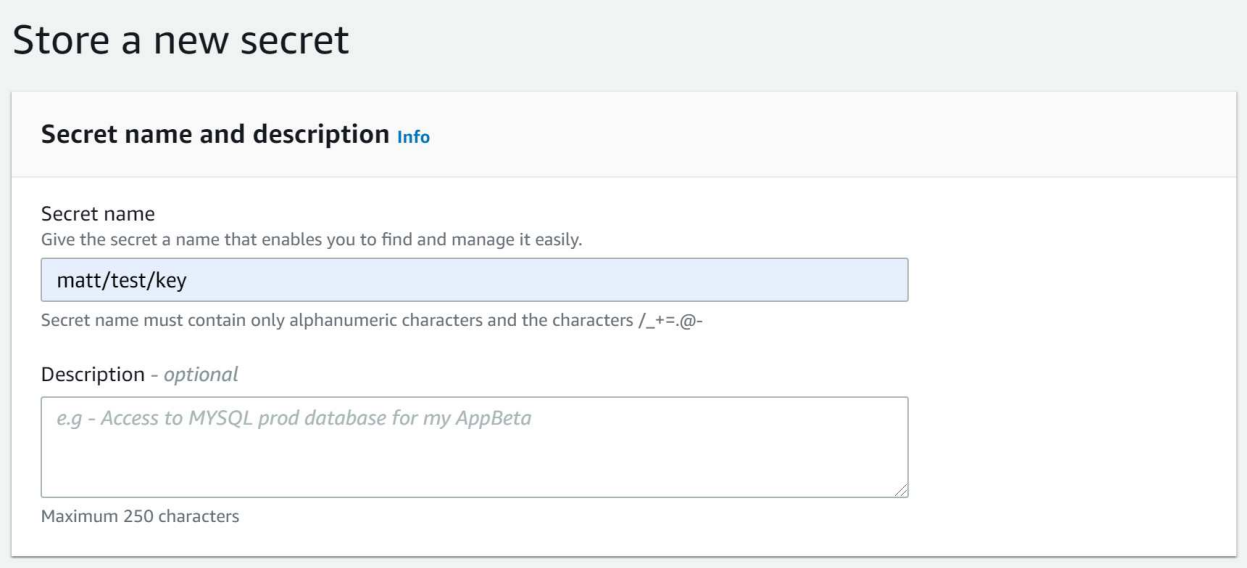
AWS Secret Creation

The AWS cloud config source takes configuration from a single named secret in a JSON format, with each object key being a config property name. To create a secret of this format, you must first go to the AWS secret manager home: https://console.aws.amazon.com/secretsmanager/home.

Once you've logged in, click the button to create a new secret. From here, you should specify the type as *Other type of secrets*. You'll be able to enter your config properties in a key/value format below, or leave blank for configuring through Payara Server.



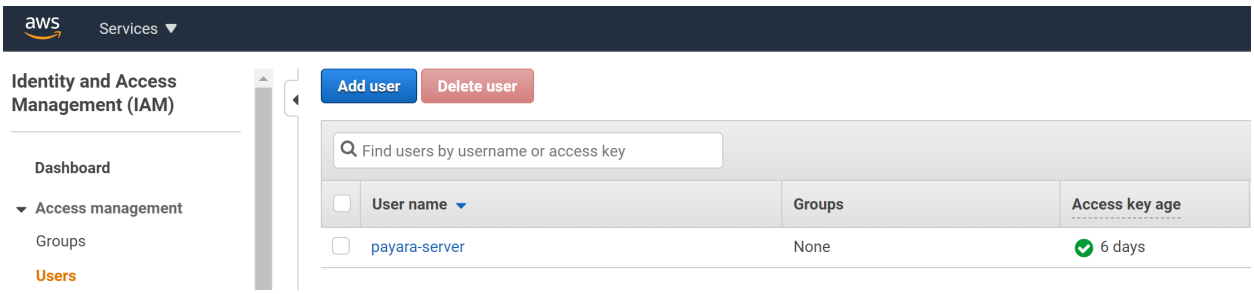
On the next screen, you'll be able to enter your secret name. This will need giving to Payara Server in order to fetch the config properties from it.



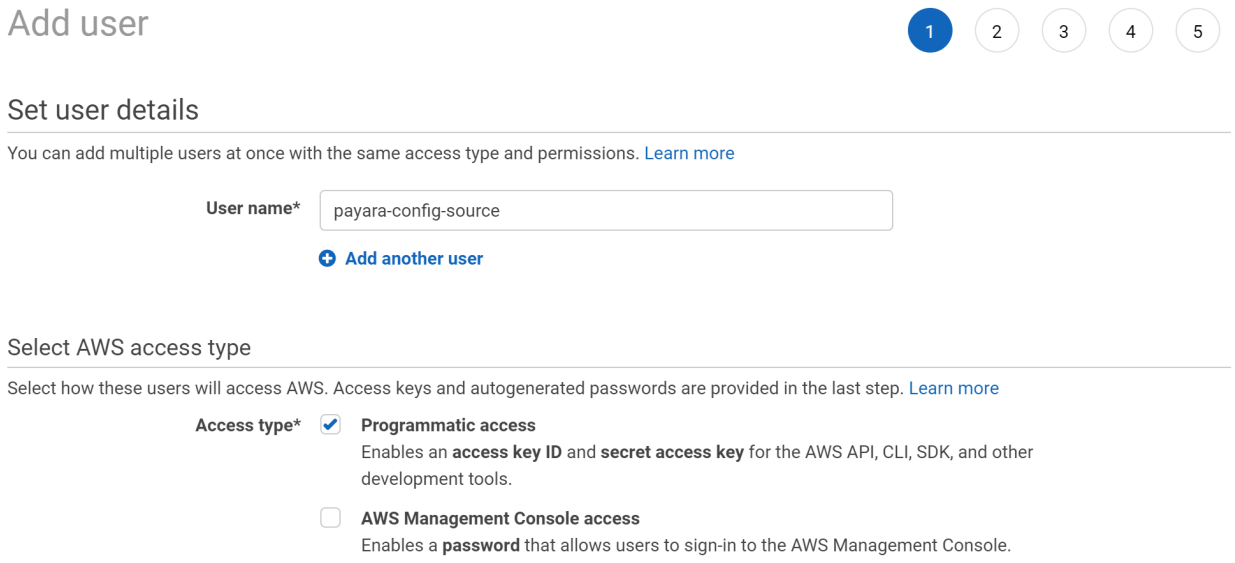
AWS IAM User

In order to connect to AWS Secrets Manager you need to know your project name as well as have an access key. Assuming you already have an AWS project and a secret created in AWS Secrets Manager, you need to create an IAM user which Payara Server will use to access your AWS Secrets.

From the AWS console, visit the *IAM Users* dashboard by searching 'IAM' in the top search bar. From here, head to *Users* in the left menu. You'll see all your IAM users associated with this region.

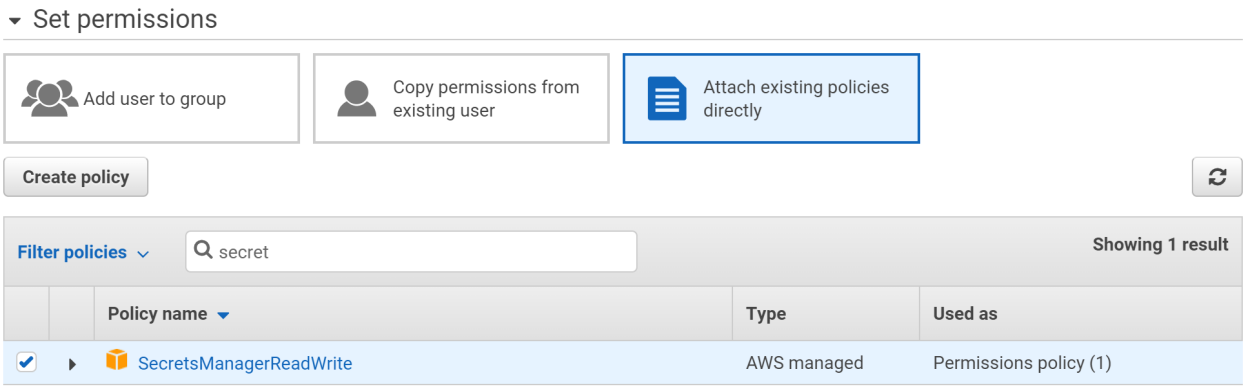


From here you can create a new IAM user. Skip this step if you already have an IAM user you want to use.



Set the IAM username. This isn't used by Payara, but will be helpful for your own reference. Make sure you enable *Programmatic access*, as this will be used by Payara Server to access your AWS Secrets.

Next, select your IAM user permissions.



Whether you assign a group or select individual permissions, make sure the IAM user contains the *SecretsManagerReadWrite* permission, which will allow the user access to your secrets.

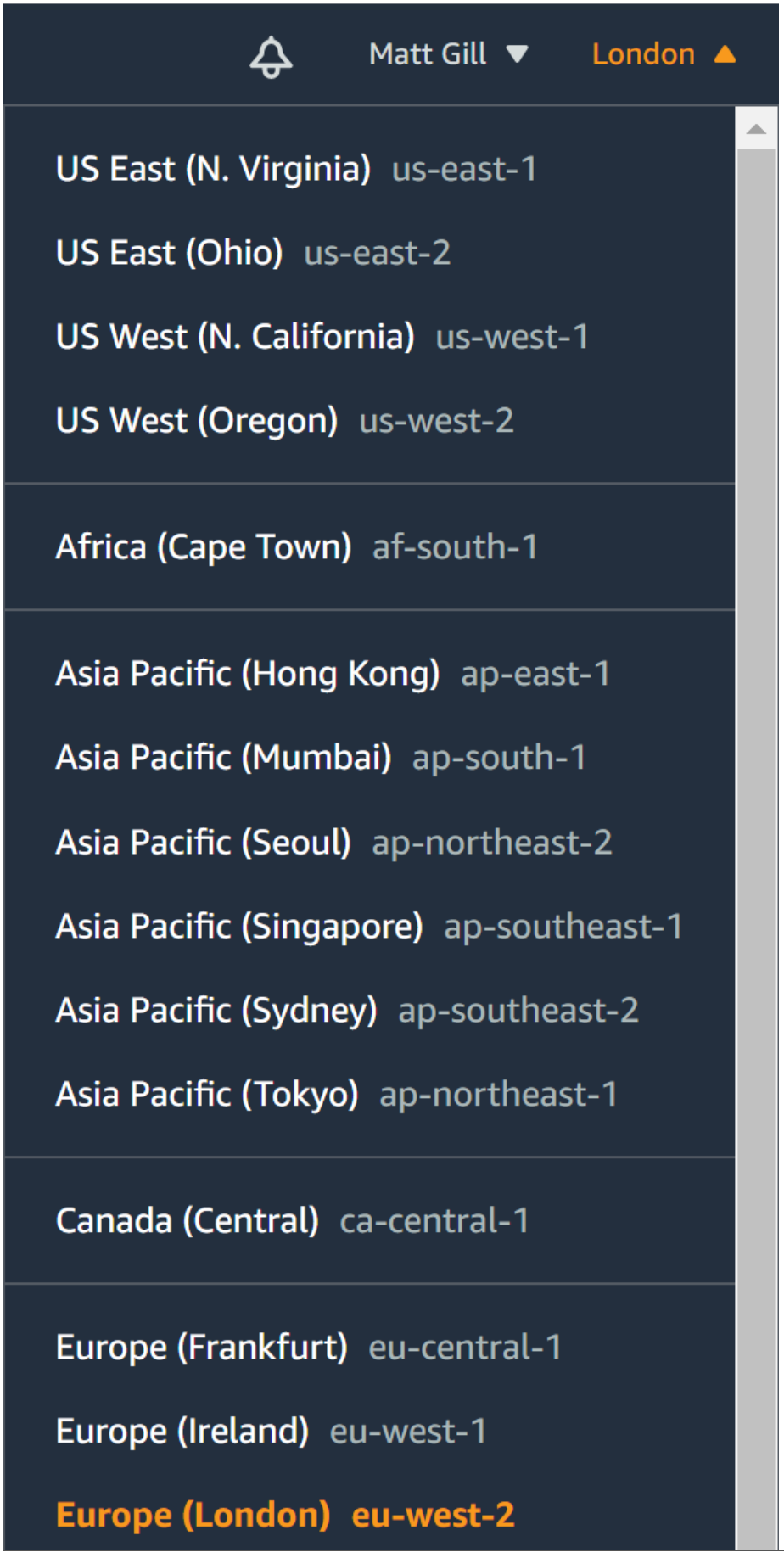
When you finish creating the IAM user, you'll be given an access key id and secret access key. These will need to be recorded, and passed to Payara Server as administering-password-aliases `AWS_ACCESS_KEY_ID` and

`AWS_SECRET_ACCESS_KEY` respectively.

Configuration

You can configure AWS Secrets either via the admin console or the *asadmin* utility. You'll need the access key id, secret access key and secret name created from the AWS console in the previous sections, as well as the name of the AWS region. Make sure that the AWS Secrets Manager has been enabled in the specified region.

To find the region in the AWS console, check the top right dropdown:


Matt Gill
London
US East (N. Virginia) us-east-1
US East (Ohio) us-east-2
US West (N. California) us-west-1
US West (Oregon) us-west-2
Africa (Cape Town) af-south-1
Asia Pacific (Hong Kong) ap-east-1
Asia Pacific (Mumbai) ap-south-1
Asia Pacific (Seoul) ap-northeast-2
Asia Pacific (Singapore) ap-southeast-1
Asia Pacific (Sydney) ap-southeast-2
Asia Pacific (Tokyo) ap-northeast-1
Canada (Central) ca-central-1
Europe (Frankfurt) eu-central-1
Europe (Ireland) eu-west-1
Europe (London) eu-west-2

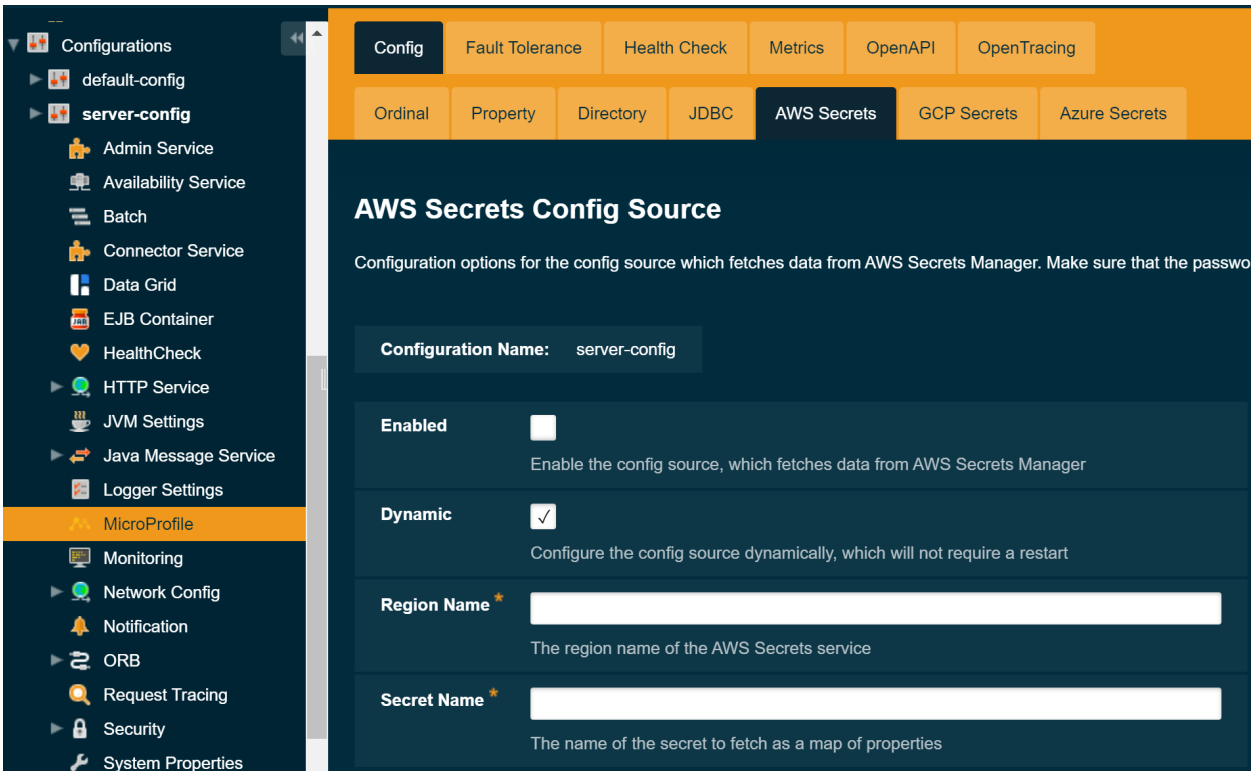
NOTE

The region name is the snake-case notation, not the full words. E.g. `eu-west-2`.

From the Admin Console

To configure the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` password aliases from the admin console, refer to the administering-password-aliases guide.

To configure the config source from the admin console, go to `Configs` → `your-config` → `MicroProfile` → `Config` → `AWS Secrets`.



From here you can pass the name of the secret to access in AWS Secrets Manager, as well as the region name. You can also decide whether to apply these changes dynamically or on the next server restart. If the config source is enabled or disabled dynamically it will take effect across the server immediately.

From the Command Line

To configure the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` password aliases from the command line, refer to the "Password Aliases Asadmin configuration"

guide.

To configure the AWS Secrets Cloud Config Source from the command line, use these commands:

- set-aws-config-source-configuration
- get-aws-config-source-configuration

Usage

Provided the required roles have been assigned to the IAM user in the AWS console, the secrets can be injected into any applicable MicroProfile Config injection point as with any other configuration source.

The secrets can also be fetched, created and deleted from the `asadmin` utility.

To fetch a secret from AWS Secrets Manager:

```
asadmin get-config-property --source cloud --sourceName aws --propertyName mysecret

secretvalue
```

To create or change a secret from AWS Secrets Manager:

```
asadmin set-config-property --source cloud --sourceName aws --propertyName mysecret
--propertyValue secretvalue
```

To delete a secret from AWS Secrets Manager:

```
asadmin delete-config-property --source cloud --sourceName aws --propertyName
mysecret
```

Azure Cloud Config Source

The Azure cloud config source takes configuration properties from Azure Key Vault Secrets.

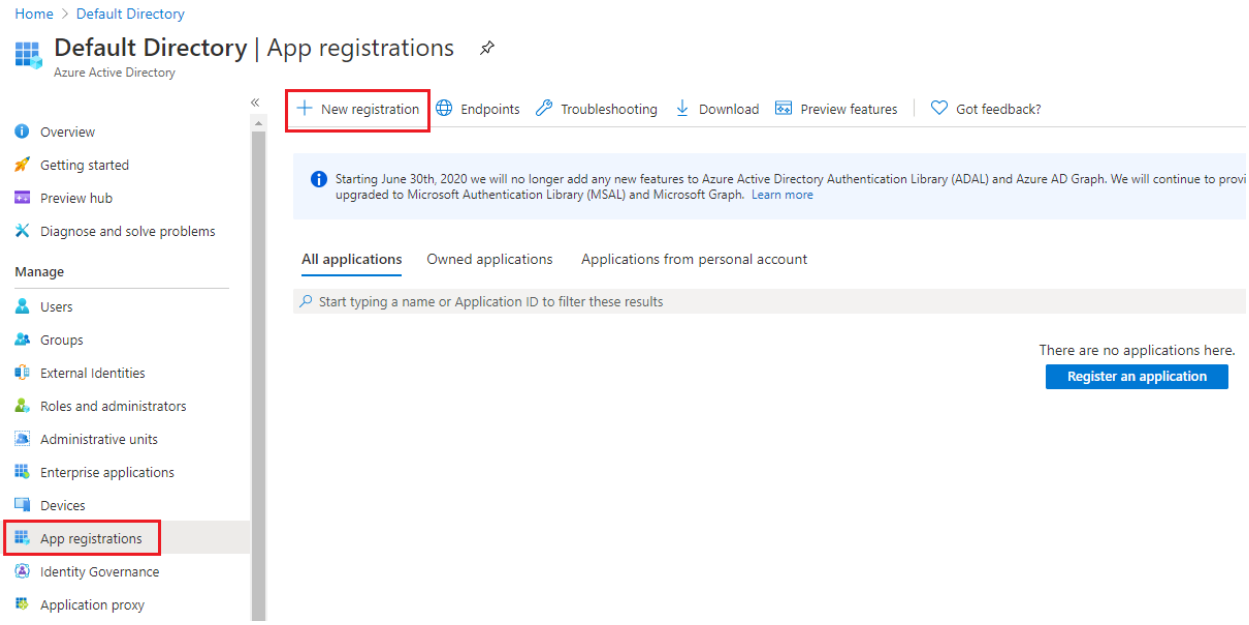
Register Azure AD Application and Create Service Principle

If you haven't already, you will need to register an Azure Active Directory (Azure AD)

application and create a service principal, so that the Microsoft identity platform can provide authentication and authorisation services. You can use the Azure portal to register your application and create a service principle.

1. Once you have signed in to your Azure portal, search and select Azure Active Directory
2. Select App registrations under the Manager section
3. Click New registration

> **NOTE** If you have access to multiple tenants and want to select a specific tenant to register with your application, use the Directory + subscription filter in the top menu.

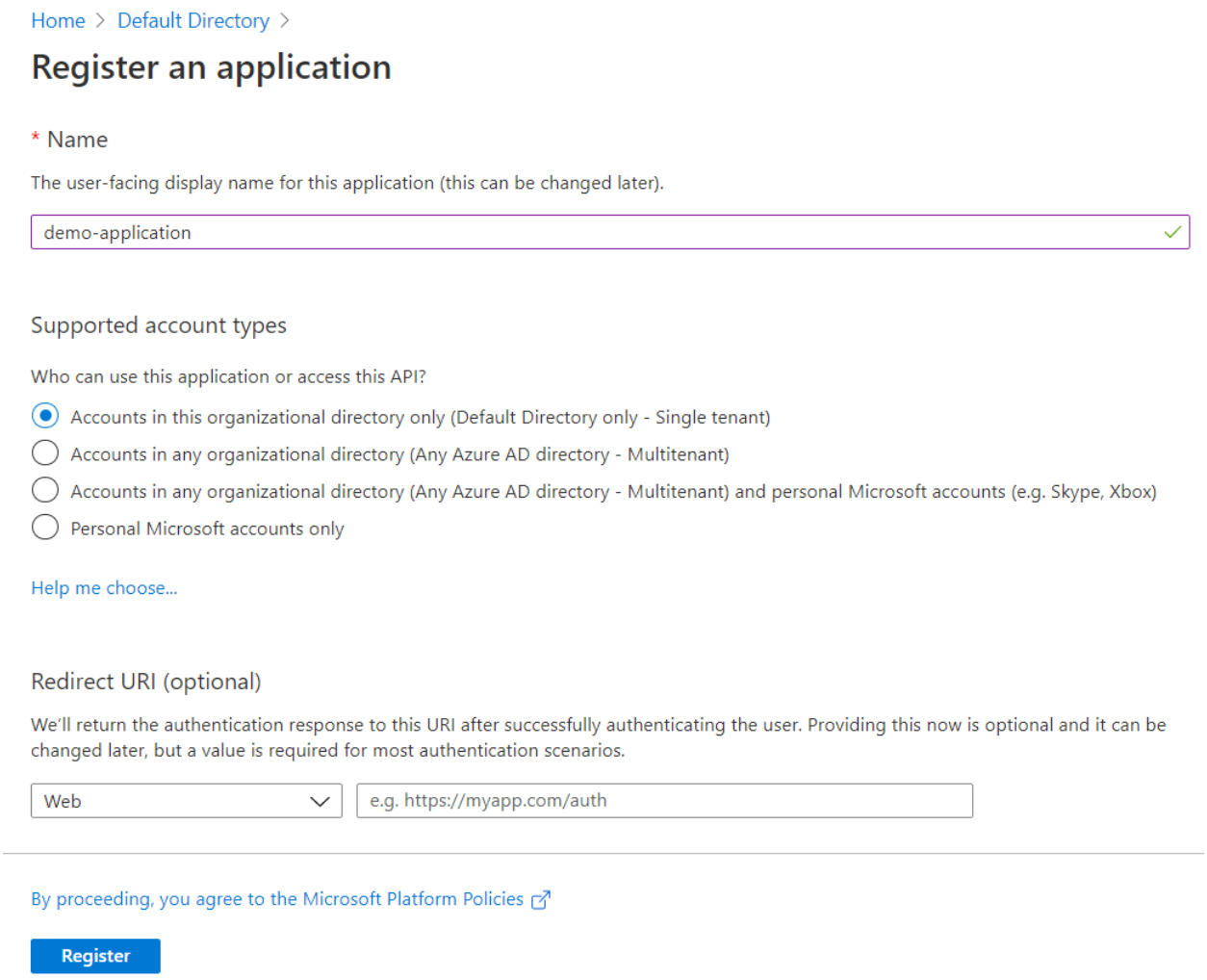


Then on the *Register an application* page:

1. Specify a name for your application
2. Select the appropriate account types
3. Click Register to complete your application registration

Once the application registration is completed, the service principal is automatically

created in your home tenant.



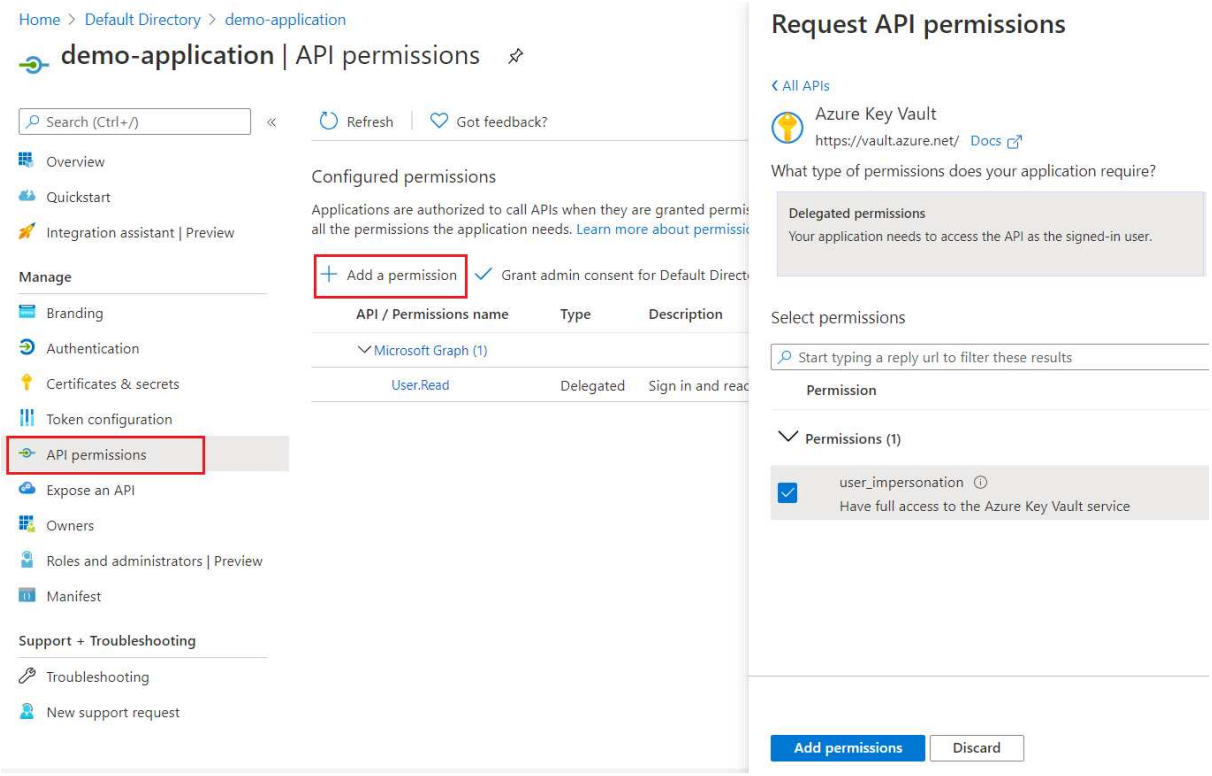
Application API Permissions

You will need to allow your application full access to the Azure Key Vault service to enable it to retrieve secrets from the vault. To do this, you will need to configure your application's API permissions.

You can learn more about permissions and consent here.

1. Select your application
2. Select API permissions under the Manage section
3. Click Add a permission
4. Select Azure Key Vault for the list of Microsoft APIs

5. Under the types of permissions, select Delegated permissions
6. Tick the *user_impersonation* checkbox
7. Click Add permissions



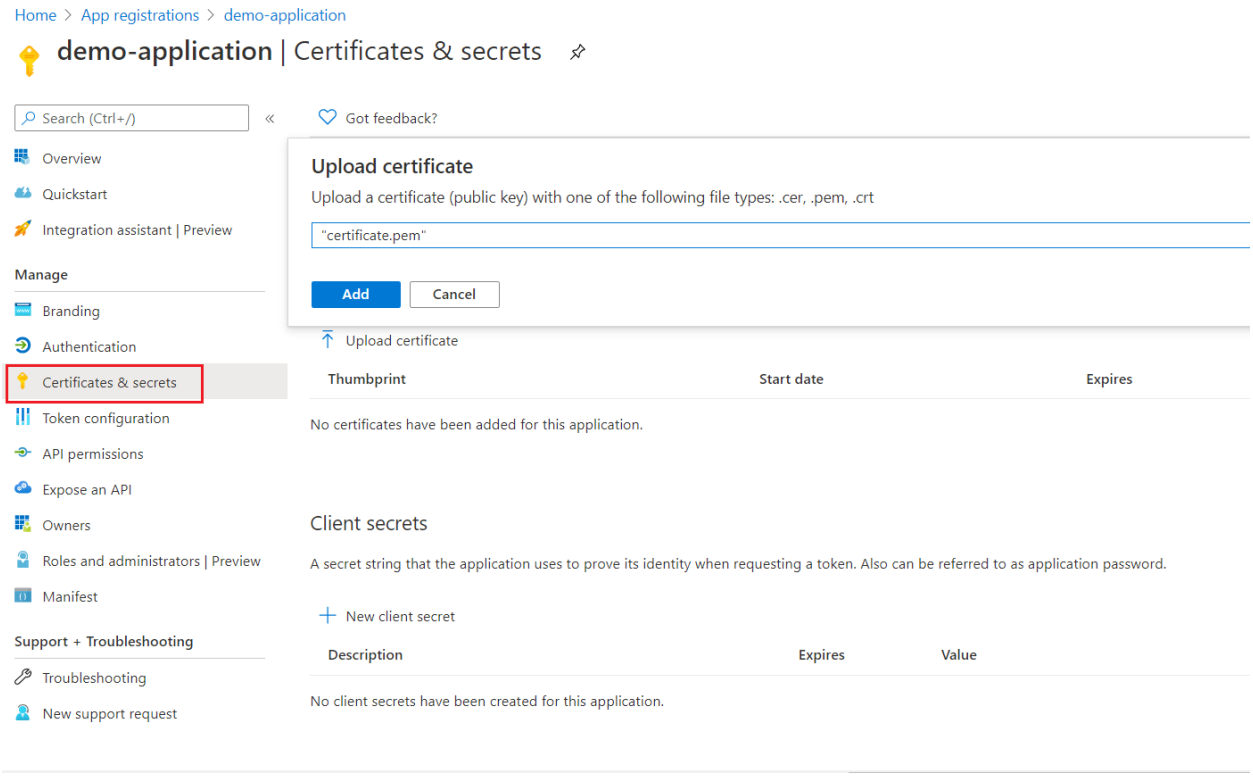
Adding Credentials

Credentials allow your application to authenticate as itself. Azure supports two options as credentials: client secret (a string) and certificate (public key). Currently, Azul Payara only supports certificate as they provide a higher level of assurance than client secrets.

You can upload your certificate for authentication using the following steps:

1. Select your application
2. Select Certificates & Secrets under the Manage section
3. Click Upload certificate
4. Choose a certificate
5. Click *Add*

Once the certificate is successfully uploaded, it should generate a *Thumbprint*, which will be required when we configure the *Azure Config* Source.

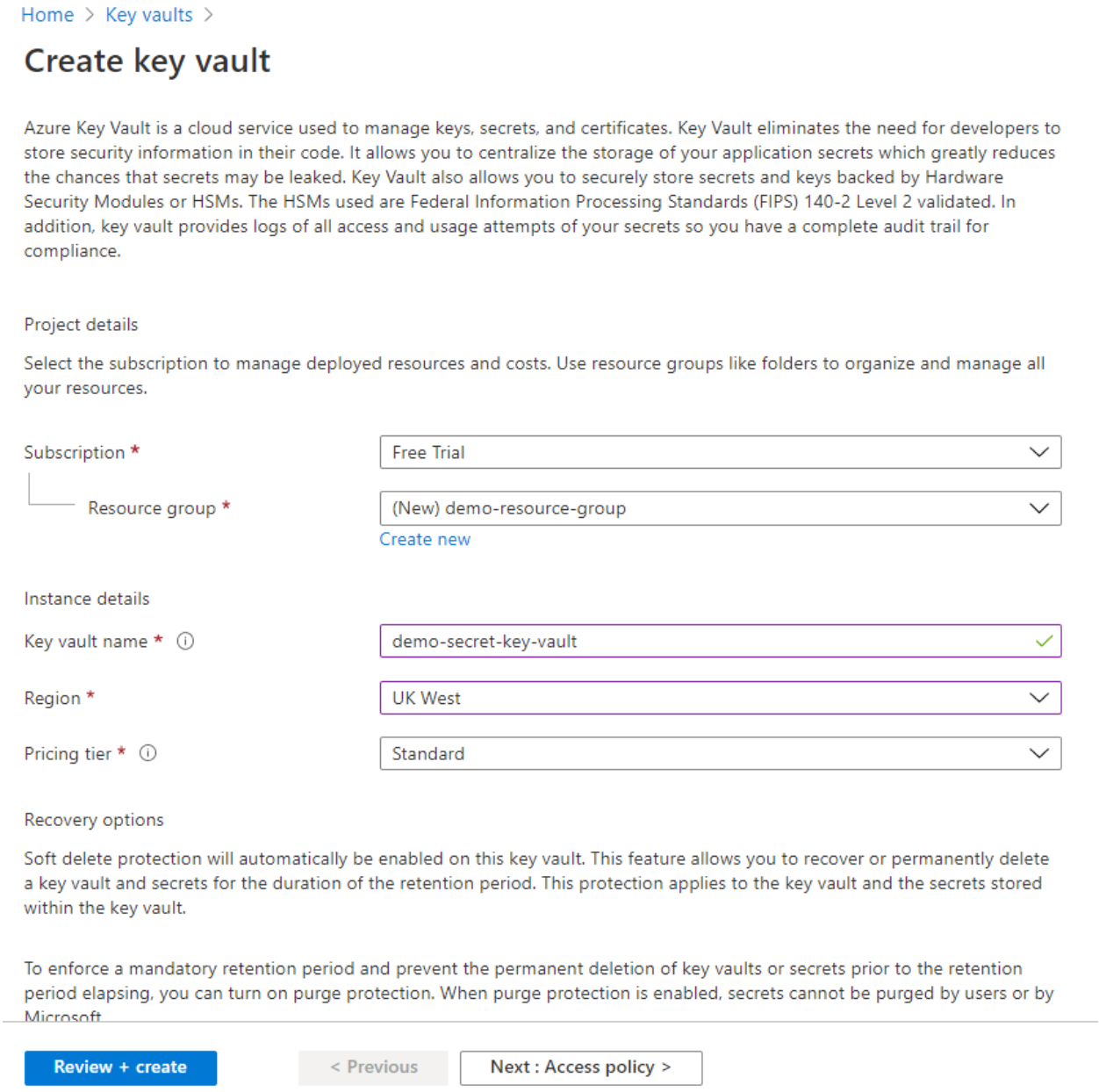


Create a Key Vault

To store your secrets securely, you will need to create a key vault.

1. Search for key vaults and select it
2. Click Add
3. Provide relevant information on the Create key vault page
4. Click *Review + create*

> **NOTE**
>
> Although key vault allows storing of keys, secrets, and certificates, *Azure Config Source* currently only support secrets.



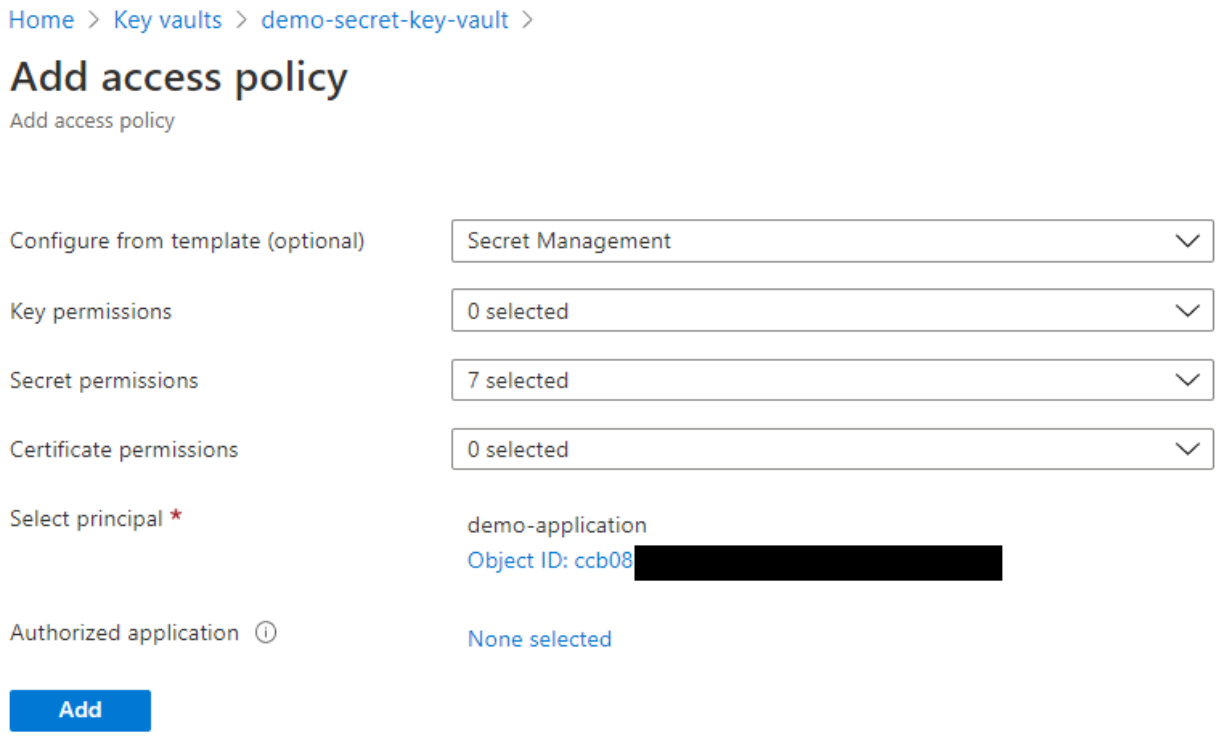
In the above diagram, the Basic section has been configured and the remaining sections have been left to their default settings. You can configure other sections, such as Access policy and Networking, to fit your requirements.

Configure Access Policies on Key Vault

To give your application access to the secrets in a key vault, you will need to add the application to the key vault's access policies.

1. Navigate to your key vault

2. Select Access policies under the Settings section
3. Click Add Access Policy
4. Under Configure from template, select Secret Management, as we don't require the other options
5. Select the service principal that you created previously
6. Click *Add*
7. Click *Save* to commit your changes

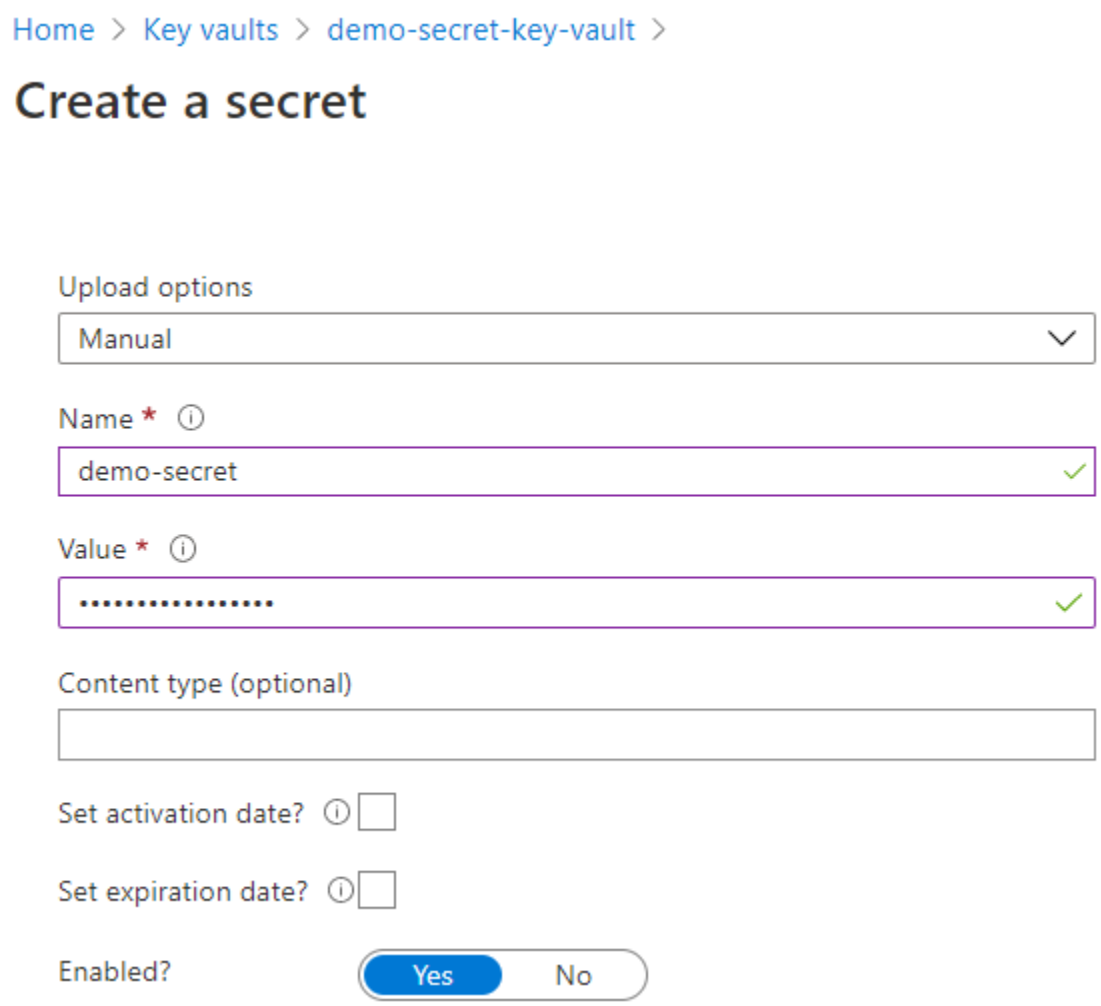


Add a Secret to Key Vault

To add a secret to the key vault:

1. Select the key vault you want to add your secret to
2. Click on Secrets under the Settings section
3. Click Generate/Import
4. Provide relevant information on the Create a secret page

5. Click Create



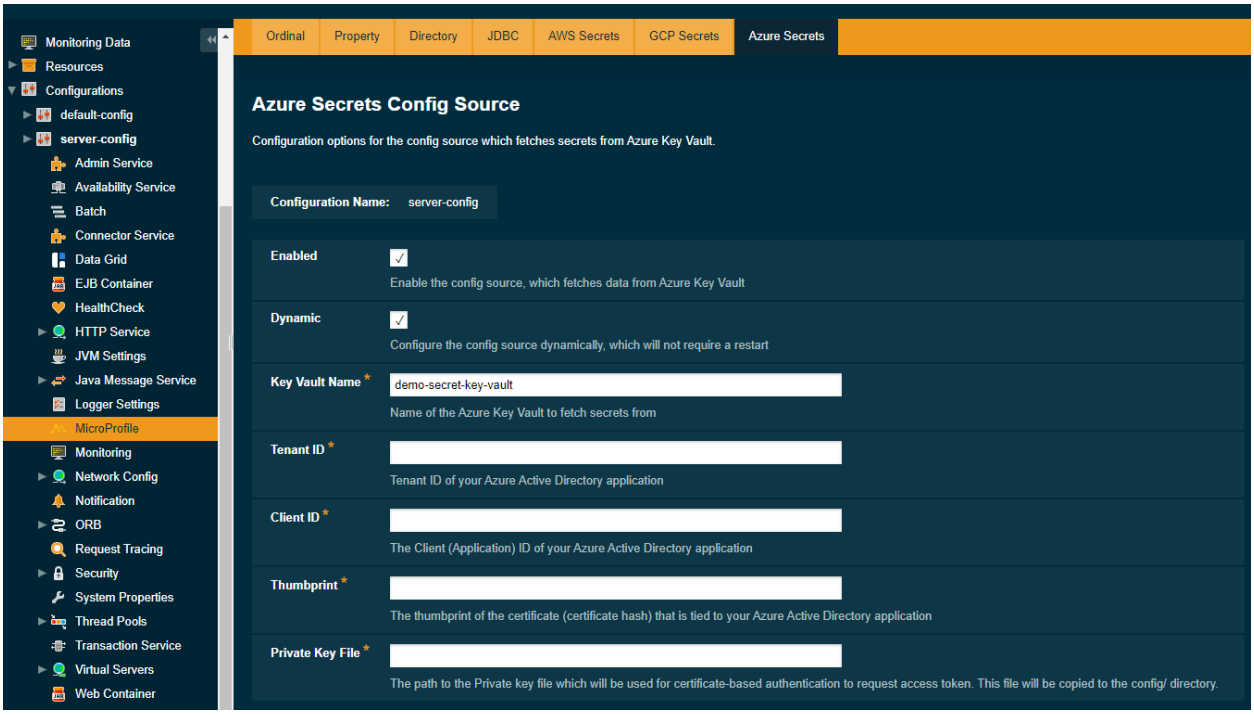
Configuration in Payara Server

You can configure Azure Secrets either via the admin console or the asadmin utility. You will need the name of your key vault, tenant ID and client ID of your application, your private key file and thumbprint generated by your application after you uploaded your certificate.

NOTE | Your private key file will be copied into `domain-dir/config`.

From the Admin Console

To configure the config source from the admin console, go to `Configs` → `your-config` → `MicroProfile` → `Config` → `Azure Secrets`.



From here, you can pass the name of the key vault, tenant ID and client ID of the application, absolute path to the private key file and thumbprint of the certificate.

You can also decide whether to apply these changes dynamically or on the next server restart. If the config source is enabled or disabled dynamically, it will take effect across the server immediately.

From the Command Line

To configure the Azure Config Source from the command line, use these commands:

- set-azure-config-source-configuration
- get-azure-config-source-configuration

Usage

Provided that all of the above sections are configured correctly, the secrets can be injected into any applicable MicroProfile Config injection point as with any other Config Source.

In Payara Server, these secrets can be fetched, created and deleted by using the `asadmin` utility.

To fetch a secret from a Key Vault:

```
asadmin get-config-property --source cloud --sourceName azure --propertyName demo-
secret
demo-secret-value
```

To create or change a secret in a Key Vault:

```
asadmin set-config-property --source cloud --sourceName azure --propertyName
mysecret --propertyValue secretvalue
```

To delete a secret from a Key Vault:

```
asadmin delete-config-property --source cloud --sourceName azure --propertyName
mysecret
```

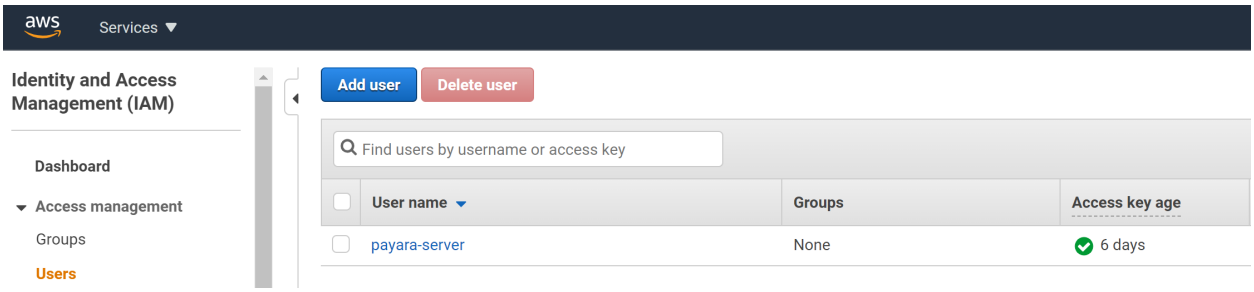
Dynamodb Config Source

The DynamoDB config source reads configuration properties from a DynamoDB database table. DynamoDB is a managed **NoSQL** database service provided by Amazon.

AWS IAM User

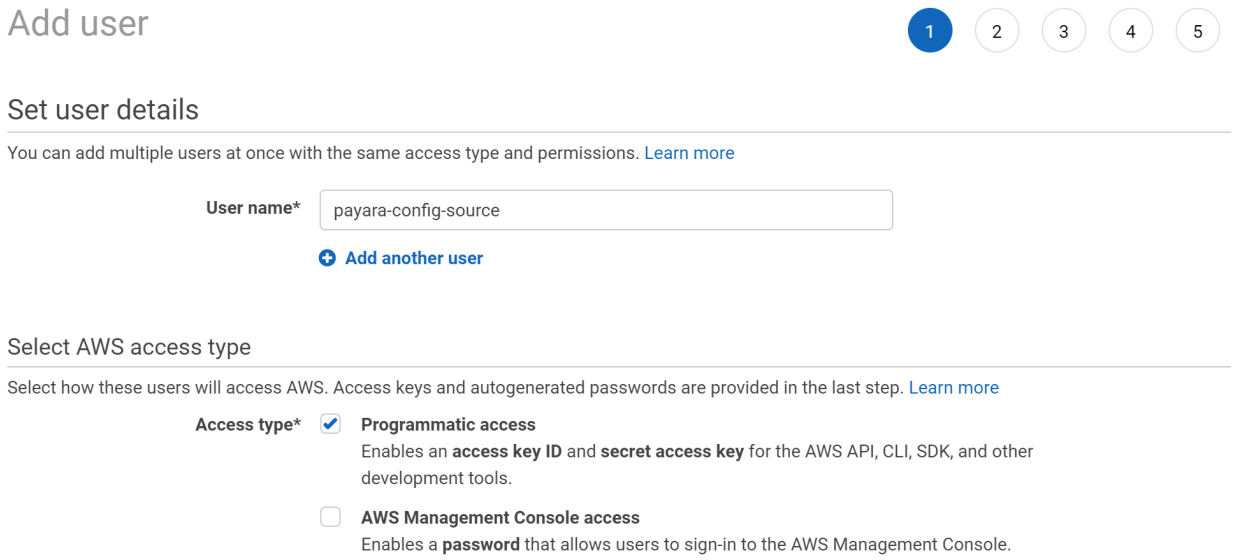
In order to access DynamoDB, you need an AWS access key. If you don't have an access key, you can create them from the AWS Management Console. To create access keys for an IAM user which Payara Server will use to access your DynamoDB.

From the AWS console, visit the *IAM Users* dashboard by searching 'IAM' in the top search bar. From here, head to *Users* in the left menu. You'll see all your IAM users associated with this region.



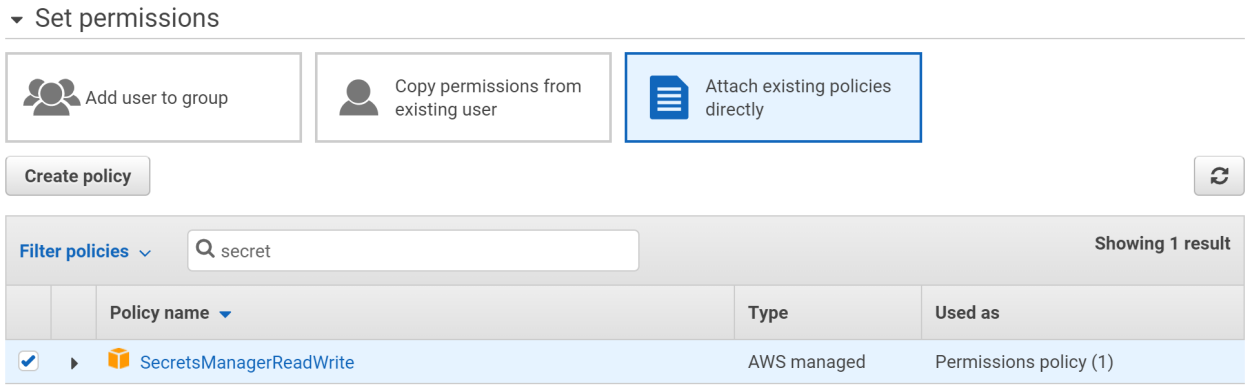
From here you can create a new IAM user. Skip this step if you already have an IAM user

you want to use.



Set the IAM username. This isn't used by Payara Server but will be helpful for your own reference. Make sure you enable *Programmatic access*, as this will be used by Payara Server to access your AWS Secrets.

Next, select your IAM user permissions.



Whether you assign a group or select individual permissions, make sure the IAM user contains the *SecretsManagerReadWrite* permission, which will allow the user access to your secrets.

When you finish creating the IAM user, you'll be given an access key id and secret

access key. These will need to be recorded and passed to Payara Server as administering-password-aliases `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` respectively.

Configuration in Payara Server

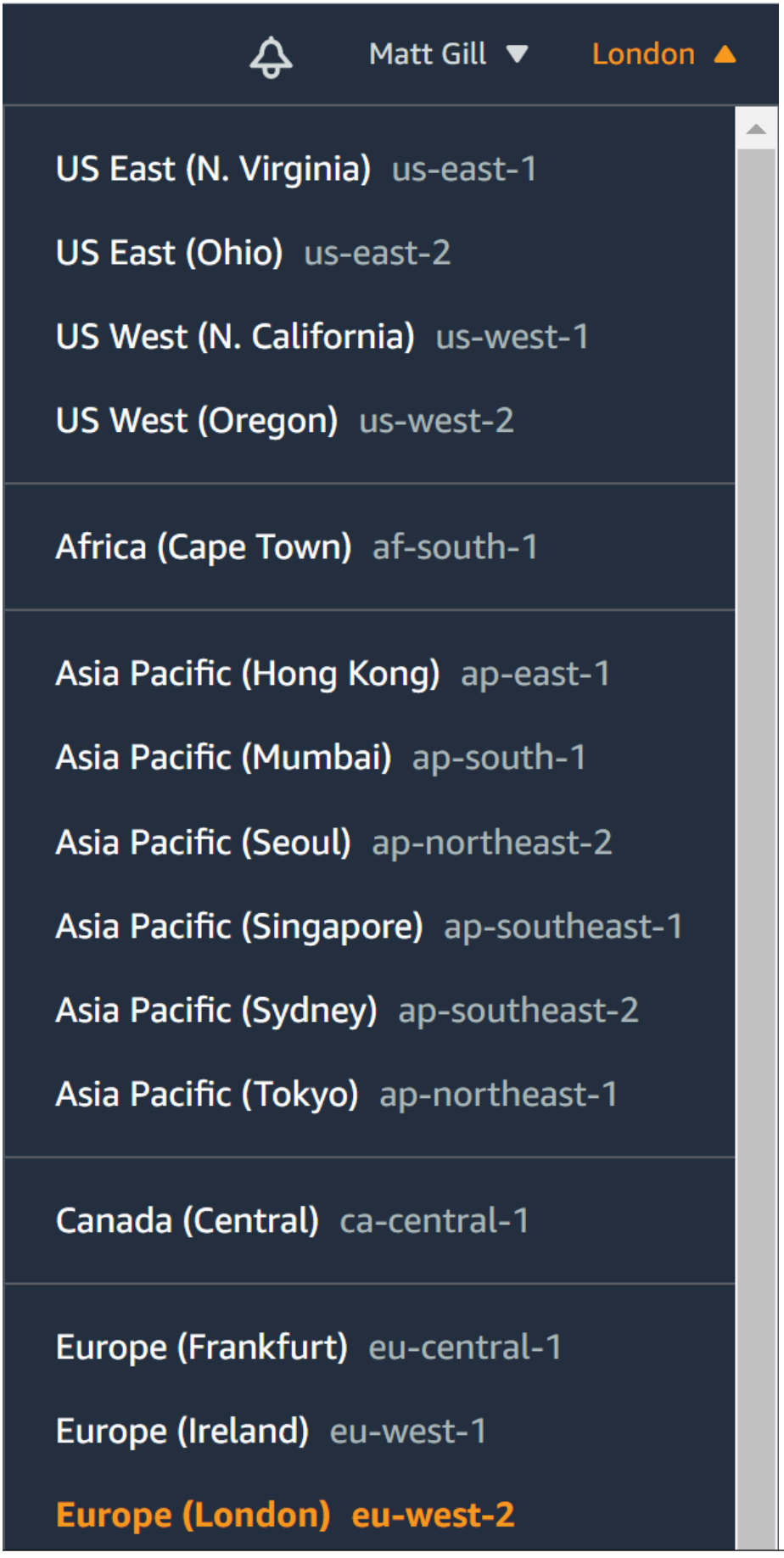
DynamoDB config source configuration is disabled by default but can be configured dynamically or statically like any other Payara service.

You can configure DynamoDB Config Source either via the admin console or the *asadmin* utility. You'll need the name of a database table, name of the key column containing the key and name of the column containing the value, as well as the name of the AWS region the DynamoDB table has been created in.

The values from the key column will be used as the config property names and the values from the value column will be used as the config property value.

IMPORTANT	The type of the database columns for the `key` and `value` columns needs to be of type `String` or compatible.

To find the region in the AWS console, check the top right dropdown:


Matt Gill
London
US East (N. Virginia) us-east-1
US East (Ohio) us-east-2
US West (N. California) us-west-1
US West (Oregon) us-west-2
Africa (Cape Town) af-south-1
Asia Pacific (Hong Kong) ap-east-1
Asia Pacific (Mumbai) ap-south-1
Asia Pacific (Seoul) ap-northeast-2
Asia Pacific (Singapore) ap-southeast-1
Asia Pacific (Sydney) ap-southeast-2
Asia Pacific (Tokyo) ap-northeast-1
Canada (Central) ca-central-1
Europe (Frankfurt) eu-central-1
Europe (Ireland) eu-west-1
Europe (London) eu-west-2

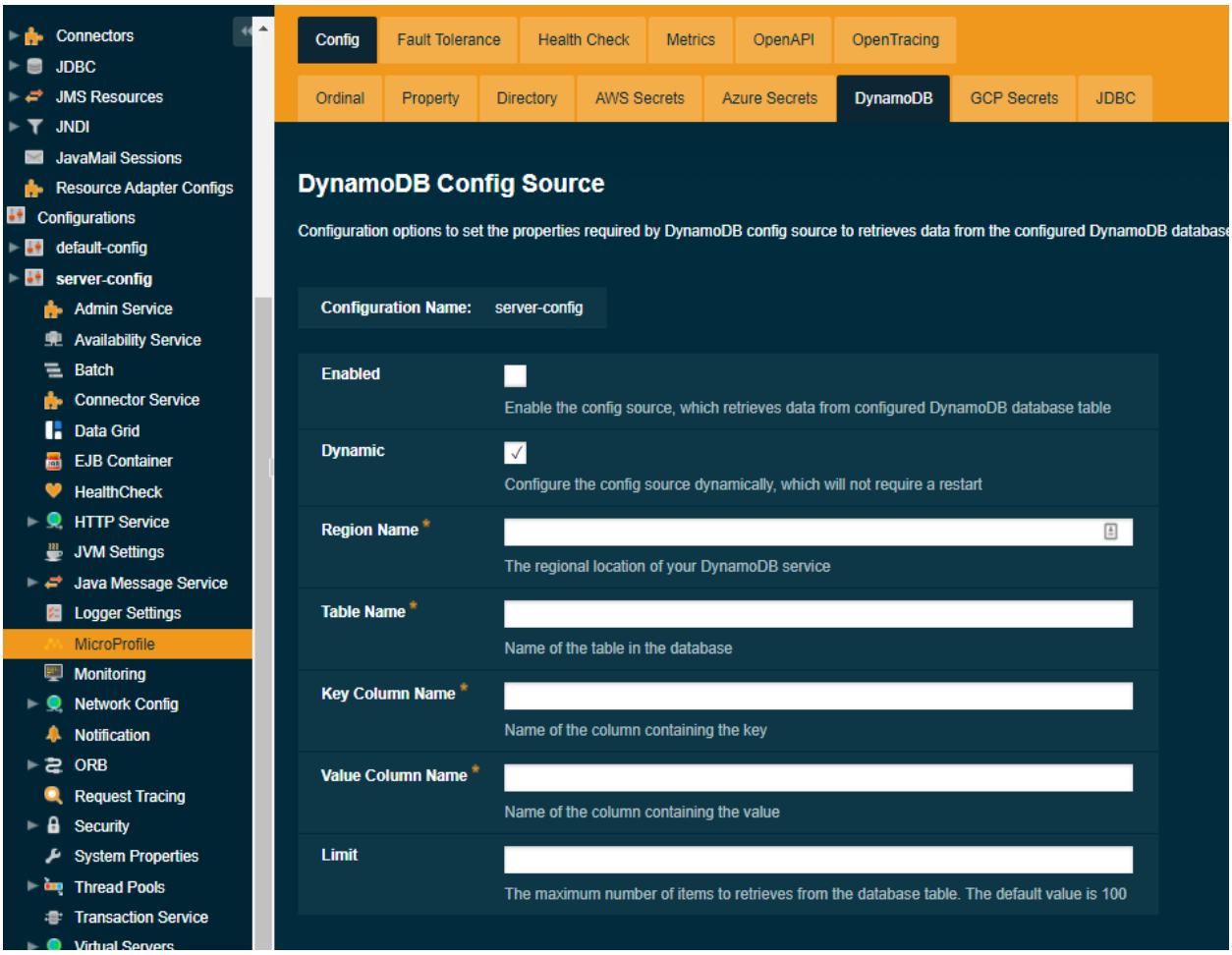
NOTE

The region name is the snake-case notation, not the full words. E.g. `eu-west-2`.

From the Admin Console

To configure the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` password aliases from the admin console, refer to the administering-password-aliases guide.

To configure the config source from the admin console, go to `Configs` → `your-config` → `MicroProfile` → `Config` → `DynamoDB`.



From here you configure the name of the database table, the region name, name of the key column containing the key, name of the column containing the value and the maximum number of items to retrieves from the database table.

From the Command Line

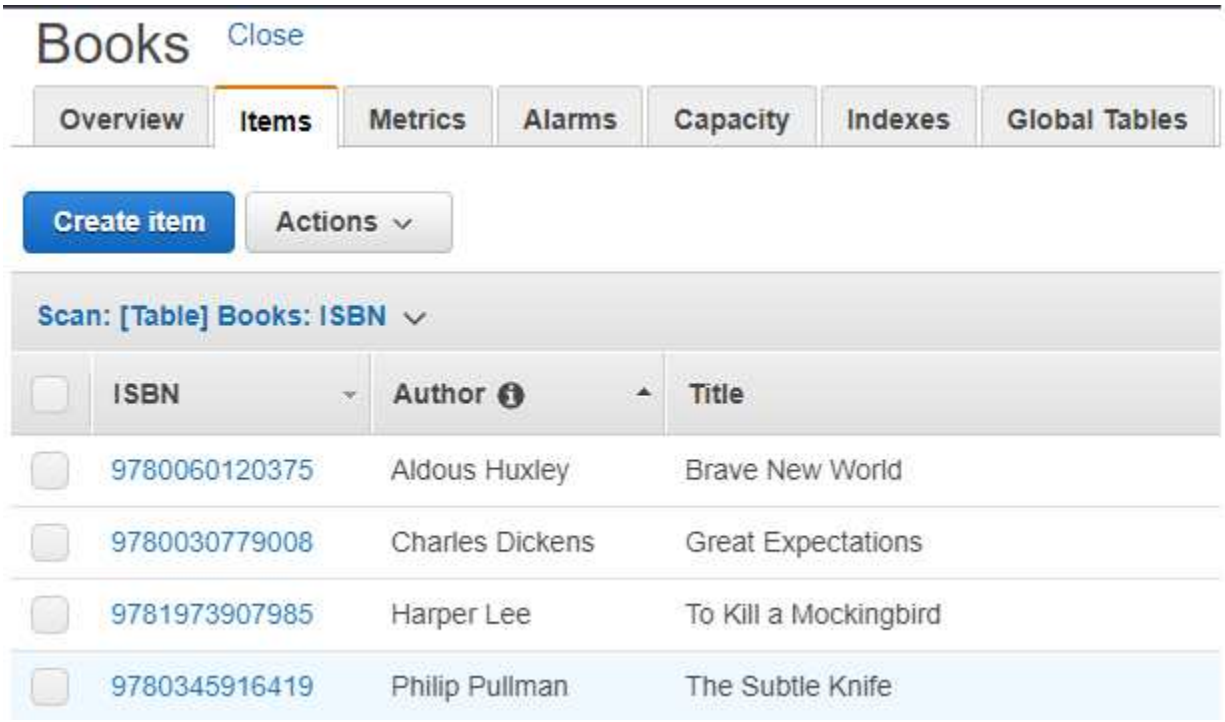
To configure the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` password aliases from the command line, refer to the administering-password-aliases guide.

To configure the DynamoDB config source from the command line, use these commands:

- set-dynamodb-config-source-configuration.
- get-dynamodb-config-source-configuration.

Usage

Once all required options are configured, you should be able to read configuration properties from a database table. For example, let's look at the following diagram. It shows a table named `Books` with columns ISBN, Author and Title.



If the key column name is set to ISBN and the value column name to Title. You can get the value of the Title column in relation to the ISBN column using the following command along with the ISBN value:

```
asadmin get-config-property --source cloud --sourcename dynamodb --propertyName
```

```
9780345916419
The Subtle Knife
Command get-config-property executed successfully.
```

NOTE Currently, DynamoDB config source only supports the `get-config-property` command and doesn't support the `set-config-property` and `delete-config-property` commands.

GCP Cloud Config Source

The GCP (Google Cloud Platform) Cloud config source takes configuration properties from the GCP Secret Manager.

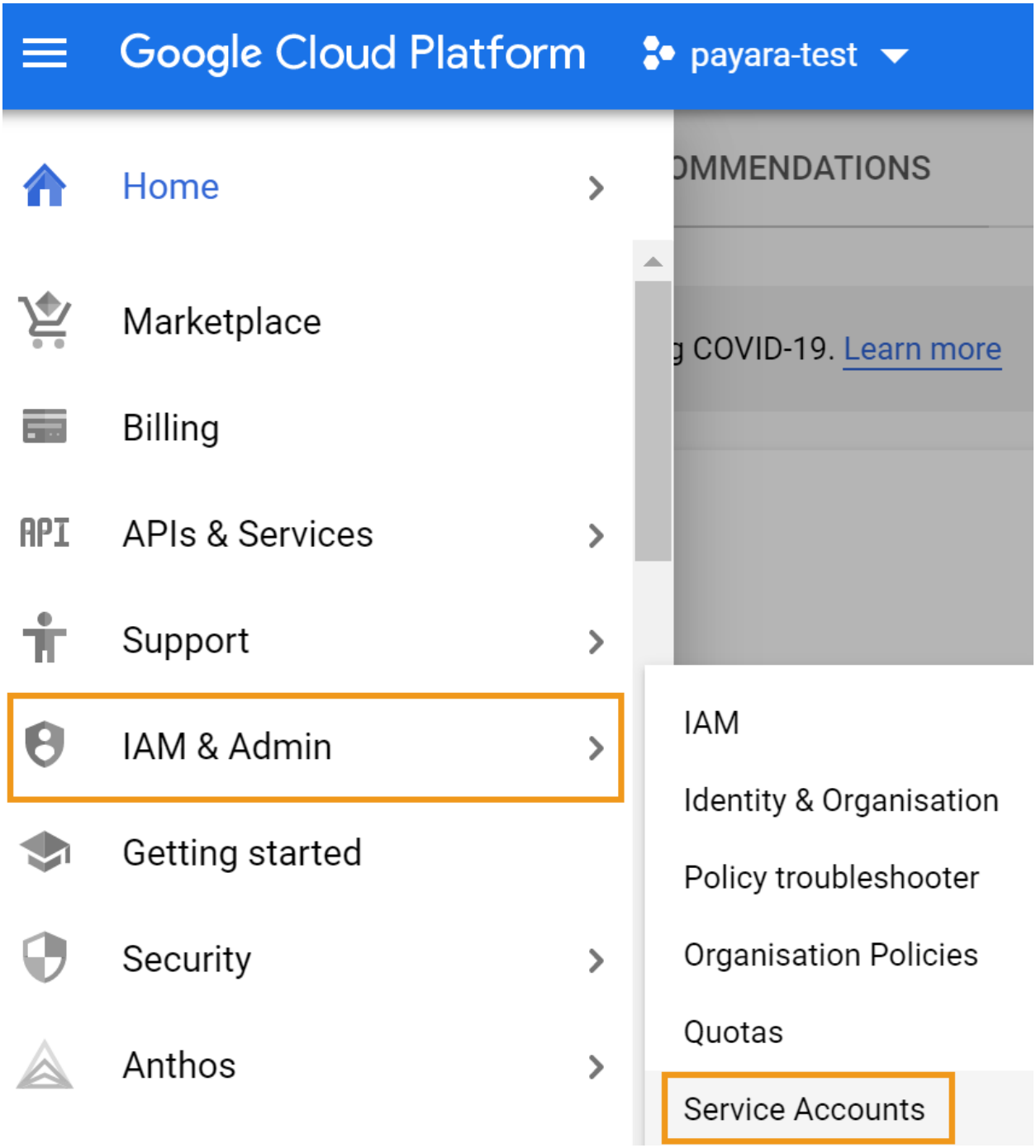
GCP Service Account

In order to connect to GCP Secret Manager you need to know your project name as well as have a service account access token. Assuming you already have a GCP project and a secret created in GCP Secrets Manager, you need to create a service account which Payara Server will use to access your GCP Secrets.

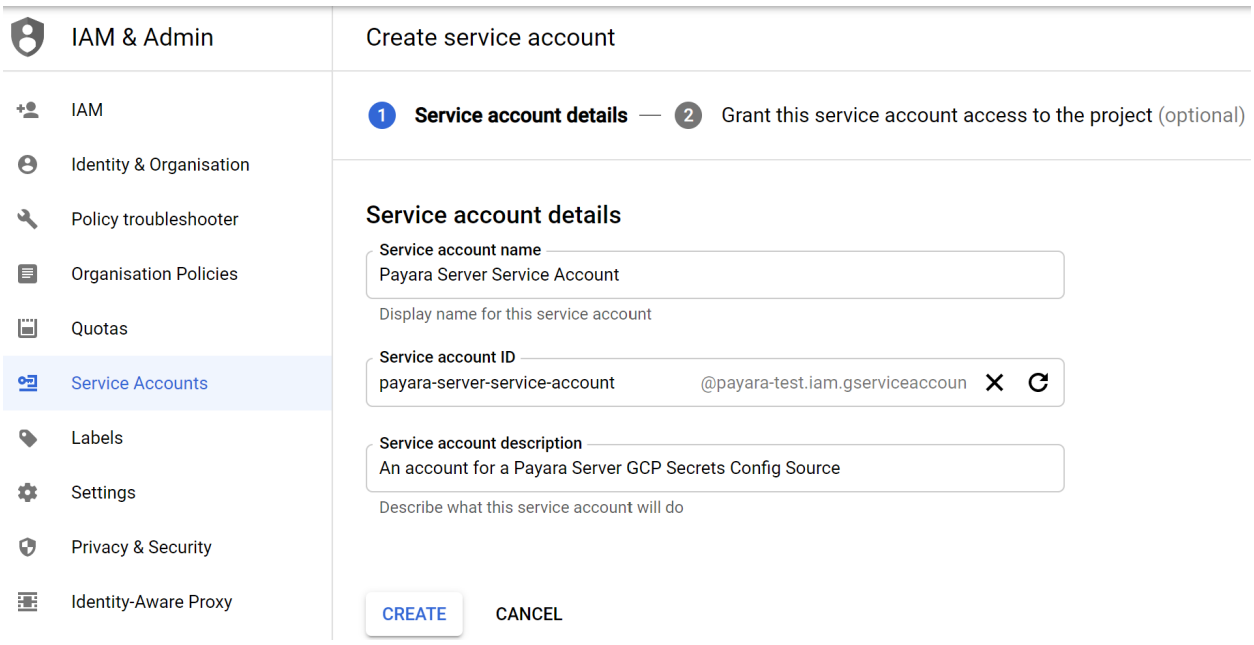
Visit the GCP cloud console: https://console.cloud.google.com/

Firstly, make sure you're on the right project. You can find your project name at the top of the GCP console.

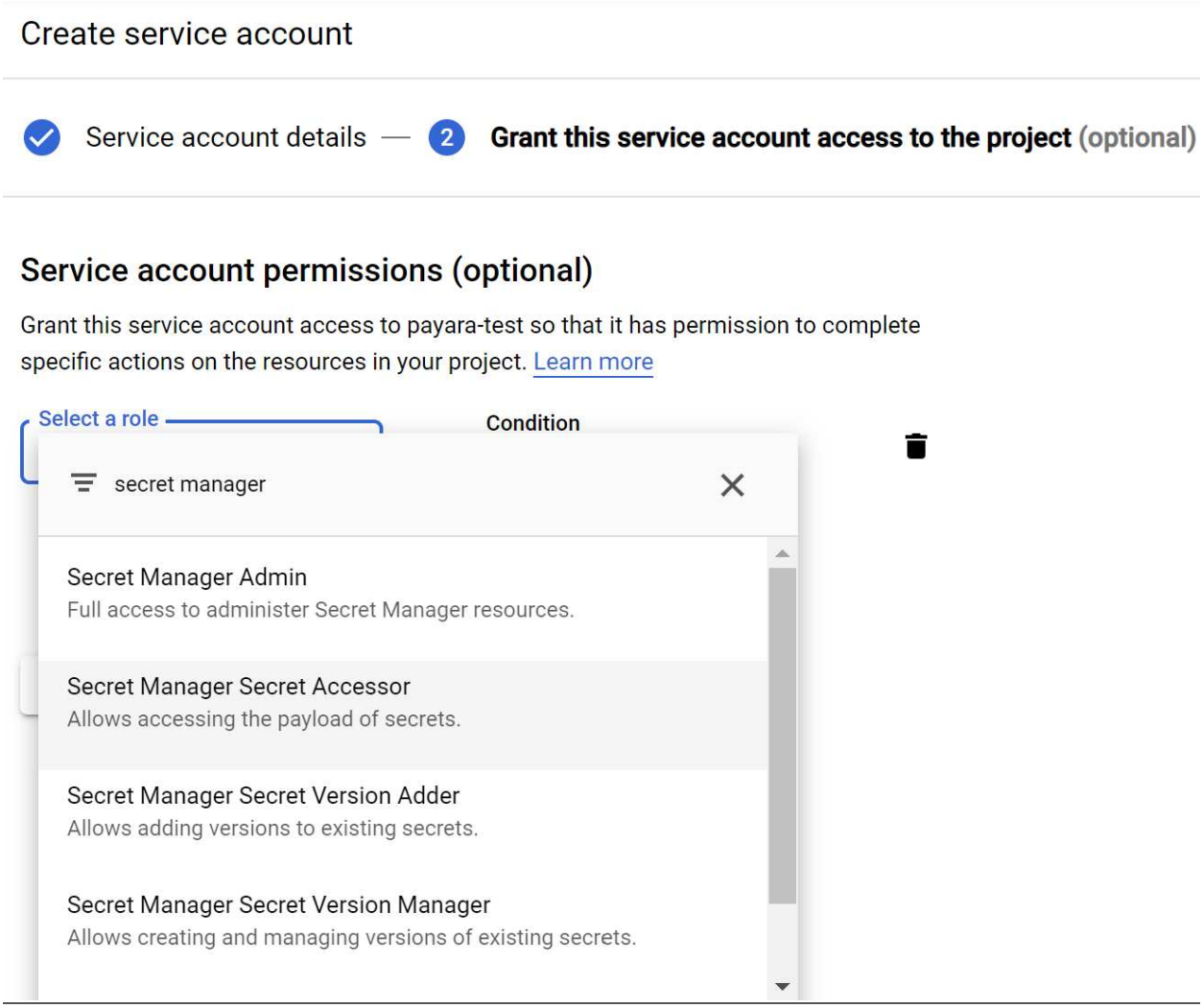
Next, visit the service account section of the project. You can find this from the top left menu ▯ *IAM & Admin* ▯ *Service Accounts*.



From here you can create a new service account. Skip this step if you already have a service account you want to use.



After this you'll need to add roles to your service account



You have a selection of roles:

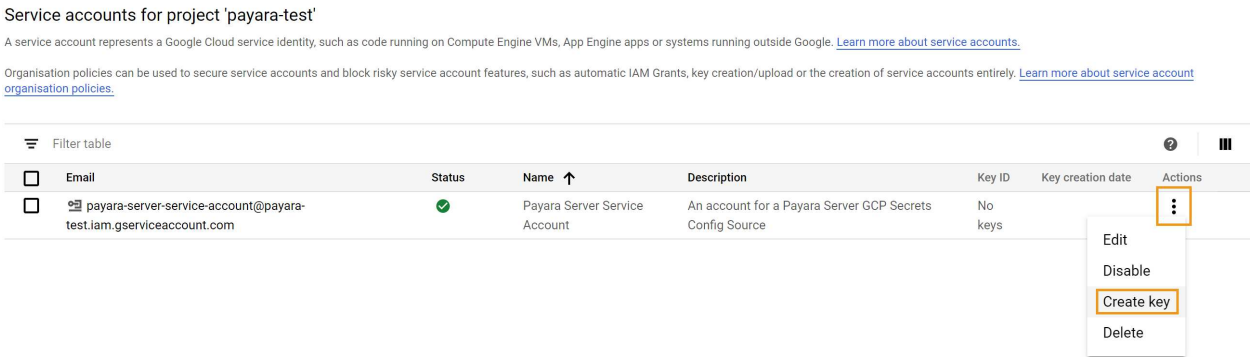
- ***Secret Manager Admin*** - full admin rights to manage secrets. Recommended when you want to delete or modify secrets through Payara Server
- ***Secret Manager Secret Accessor*** - ability to view secrets. Recommended when you only want to retrieve secrets through Payara Server
- ***Secret Manager Secret Version Adder*** - allows setting secret values
- ***Secret Manager Secret Version Manager*** - allows deleting secret values

Add whichever of these is appropriate for your use case.

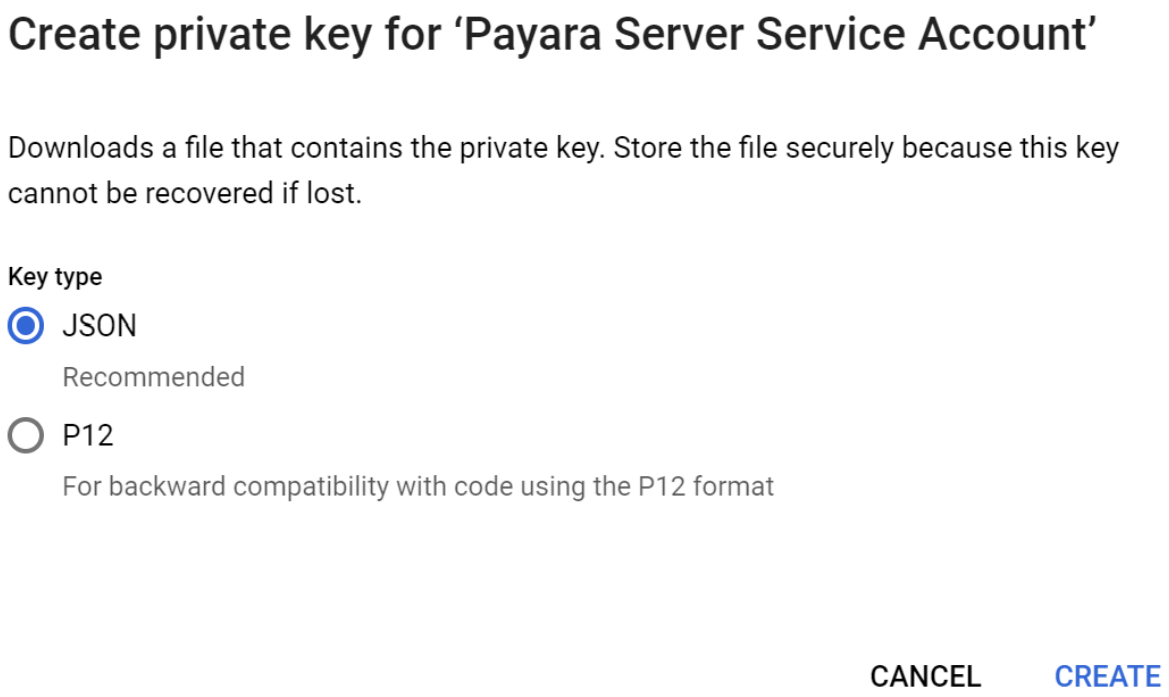
Skip step 3 if following the wizard, as no physical user will be required to use this service account.

Next, you'll need to download the access key for this service account.

Select the options icon for the service account you plan to use.



You'll be presented with an option to download the key in `JSON` or `P12` format. The GCP config source currently requires that the `JSON` format be used, so select this one.



This downloaded file can now be used by Payara Server to access GCP Secrets.

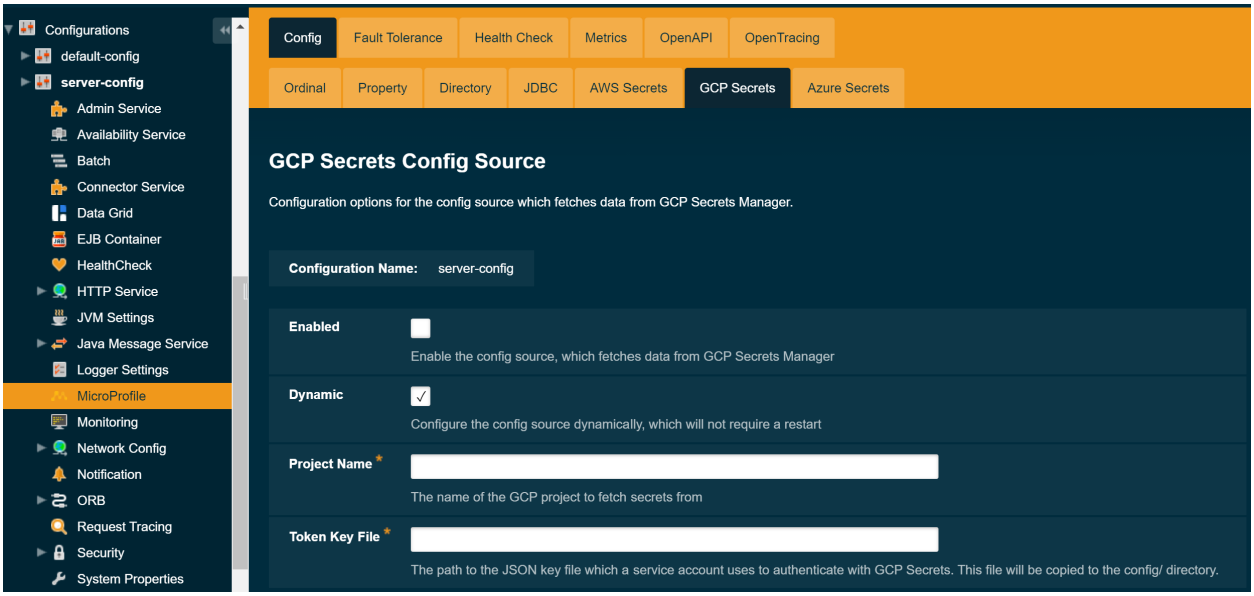
Configuration

You can configure GCP Secrets either via the admin console or the *asadmin* utility. You'll need the JSON key file created from the GCP console in the previous section, as well as the name of the project that you intend to use. Make sure that the GCP Secret Manager has been enabled for the specified project.

NOTE	The JSON key file will be copied into the Payara Server `config/` directory, so it isn't required after passing it to the GCP Config Source configuration.

From the Admin Console

To configure the config source from the admin console, go to `Configs` ▸ `your-config` ▸ `MicroProfile` ▸ `Config` ▸ `GCP Secrets`.



From here you can pass the absolute path to the json key file downloaded in the previous section, as well as the project name. You can also decide whether to apply these changes dynamically or on the next server restart.

If the config source is enabled or disabled dynamically it will take effect across the server immediately.

From the Command Line

To configure the GCP Secrets Cloud Config Source from the command line, use these commands:

- set-gcp-config-source-configuration
- get-gcp-config-source-configuration

Usage

Provided the required roles have been assigned to the Service Account in the GCP console, the secrets can be injected into any applicable MicroProfile Config injection point as with any other Config Source. The secrets can also be fetched, created and deleted from the `asadmin` utility.

To fetch a secret from GCP Secret Manager:

```
asadmin get-config-property --source cloud --sourceName gcp --propertyName mysecret

secretvalue
```

To create or change a secret from GCP Secret Manager:

```
asadmin set-config-property --source cloud --sourceName gcp --propertyName mysecret
--propertyValue secretvalue
```

To delete a secret from GCP Secret Manager:

```
asadmin delete-config-property --source cloud --sourceName gcp --propertyName
mysecret
```

Hashicorp Secrets Config Source

The HashiCorp Secrets Config Source takes configuration properties from HashiCorp Vault. You can learn more about HashiCorp on: https://www.vaultproject.io/

Prerequisite

This documentation assumes the following:

- You have successfully installed the HashiCorp Vault and have a running Vault server
- You are familiar with most, if not all, of the HashiCorp Vault technology and terminology. If not please read through their documentation before you attempt to use HashiCorp Secrets Config Source
- Your HashiCorp Vault is unsealed

IMPORTANT	Hashicorp config source only supports the Vault KV Secrets Engine.

Configuration in Payara Server

You can configure HashiCorp Secrets Config Source either via the admin console or the *asadmin* utility. You'll need a client token, the KV Secrets Engine API version, Vault's address represented as URL and port, the path where the KV Secrets Engine is enabled and the path where your secrets are stored.

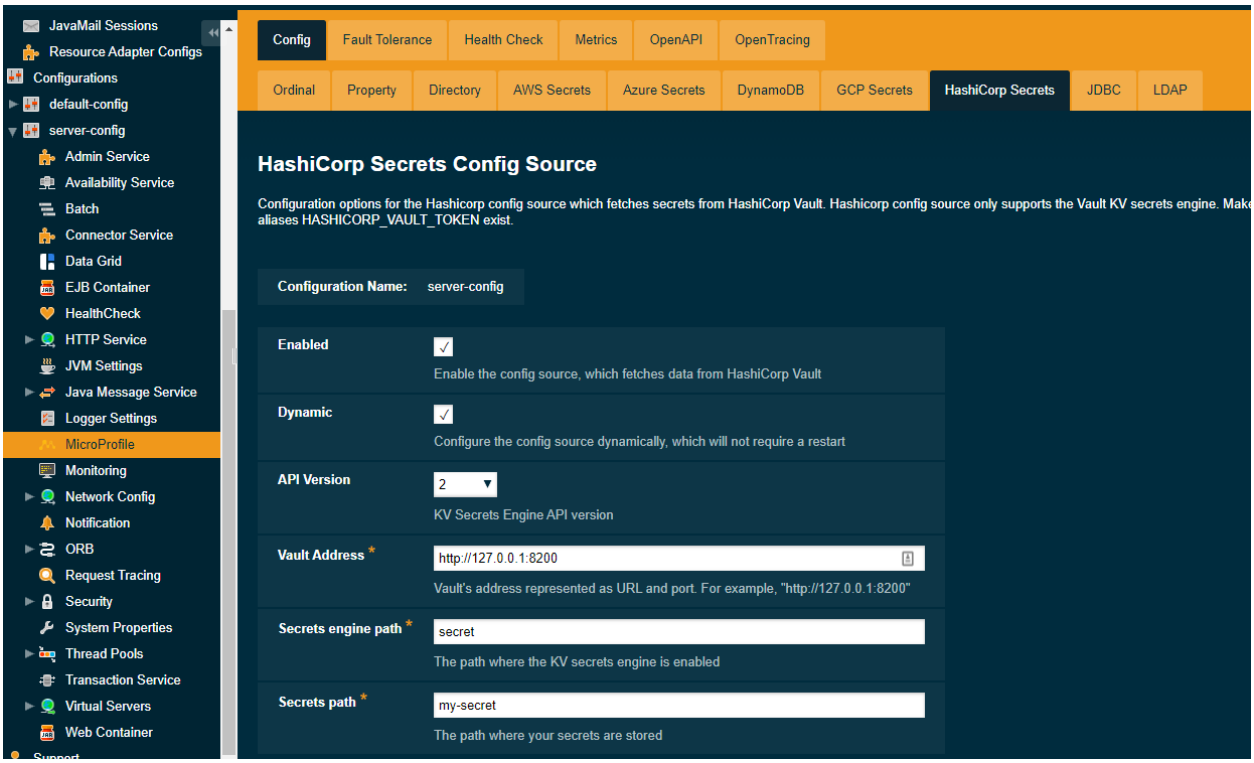
NOTE

By default, the KV Secrets Engine is enabled at the `/secret` path in Vault.

From the Admin Console

To configure the `HASHICORP_VAULT_TOKEN` password alias from the admin console, refer to the administering-password-aliases guide.

To configure the config source from the admin console, go to `Configs` → `your-config` → `MicroProfile` → `Config` → `HashiCorp Secrets`.



From here you can select the KV Secrets Engine API version you want to use, configure the vault address, the path where the KV Secrets Engine is enabled and the path where your secrets are stored.

You can also decide whether to apply these changes dynamically or on the next server restart. If the config source is enabled or disabled dynamically it will take effect across the server immediately.

From the Command Line

To configure the `HASHICORP_VAULT_TOKEN` password alias from the admin console, refer to the administering-password-aliases guide.

To configure the HashiCorp Secrets Config Source from the command line, use these commands:

- set-hashicorp-config-source-configuration
- get-hashicorp-config-source-configuration

Usage

Provided that all of the above sections are configured correctly, the secrets can be injected into any applicable MicroProfile Config injection point as with any other Config Source. The secrets can also be fetched, created and deleted using the asadmin utility.

To fetch a secret form a HashiCorp Vault:

```
asadmin get-config-property --source cloud --sourcename hashicorp --propertyname
mysecret
```

To create or change a secret from a HashiCorp Vault:

```
asadmin set-config-property --source cloud --sourcename hashicorp --propertyname
mysecret --propertyValue secretvalue
```

To delete a secret from a HashiCorp Vault:

```
asadmin delete-config-property --source cloud --sourcename hashicorp --propertyname
mysecret
```

Directory Config Source

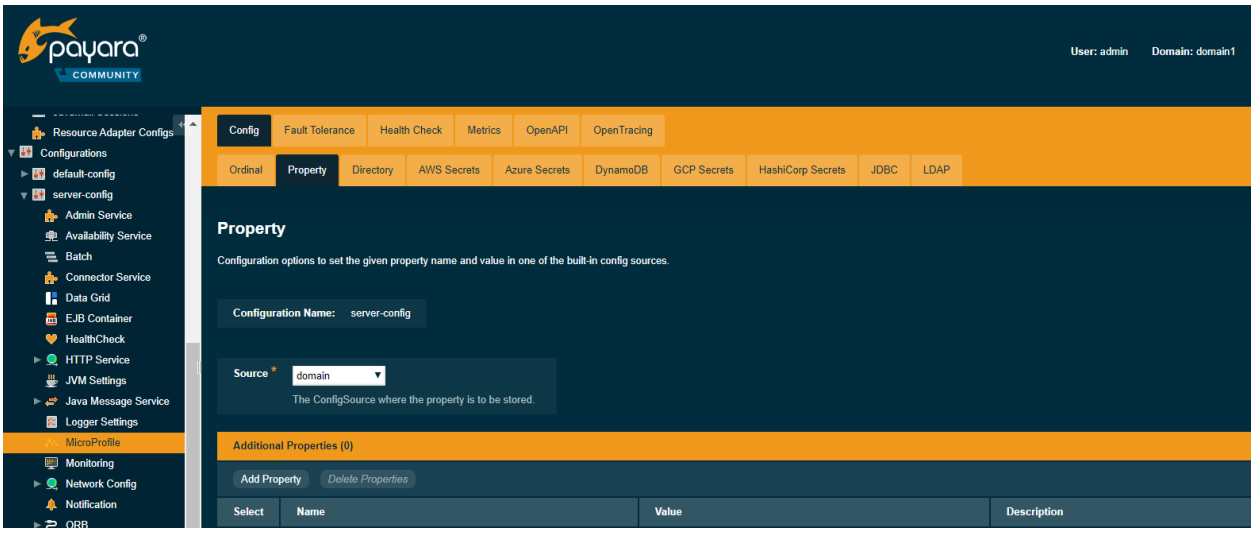
This page details how to configure and use the **Directory** configuration source in Azul Payara.

Configuration

Config Directory can be configured by using Admin Console or Asadmin commands.

Using the Admin Console

To configure the Config Directory in the Admin Console, go to Configuration [instance-configuration (like server-config)] MicroProfile Config Directory:



Using Asadmin Commands

Use the following commands to configure and get the current settings of this configuration source:

- set-config-dir
- get-config-dir

Using Pre-Boot And/or Postboot Scripts

When running an Azul Payara distribution in a container, there is no way to restart a running instance after configuring via an `asadmin` command without losing the changes to the domain configuration.

Thus, you should add the corresponding command to a boot script as described at "Pre- and Postboot scripts".

See using-environment-variables for details how to use this within non-micro containers.

Usage

Usually this config source is used to map secrets mounted by Kubernetes or Docker to a directory inside a running container.

Once you configured a directory to read a config (secrets) from, you need to make sure file names correspond to properties in your codebase.

Map Property Names to Flat File Hierarchy

1. Say you have two properties `property1` and `foo.bar.property2`.
2. Payara is configured with secret directory `/home/payara/secrets`
3. Secrets mounted as files to `/home/payara/secrets/property1` and `/home/payara/secrets/foo.bar.property2` will be read.

IMPORTANT	Remember that the files may only contain the secret or other configuration values. Plaintext files only, no file extensions allowed.

Map Property Names to a Subdirectory Structure

1. Say you have two properties `property1` and `foo.bar.property2`.
2. Payara is configured with secret directory `/home/payara/secrets`
3. Secrets mounted as files to `/home/payara/secrets/property1` and `/home/payara/secrets/foo/bar/property2` will be read.

Restrictions on the files content are the same as with the flat hierarchy.

Updates to files and subdirectories are picked up at runtime. Retrieving the config property again will use the updated values. This allows for clearing a value by removing the file, too.

Dots Usage and Depicting Directories and File Name

Dots in property names are used to reflect scopes, for example to distinguish different applications, modules, etc.

1. Any dots in the property name may correspond to changing from one directory to a subdirectory. Example: `foo.bar.test` could be a file `test` in `foo/bar/` path.
2. You may combine any number of dot-separated "components" into directories and

file name. Example: `foo.bar.test.example` may be a file `test.example` in `foo/bar/` path or a file `example` in path `foo/bar.test/` and so on.

3. Do not use a file extension, as it would be taken as part of the property name.
4. The longest, most specific match "wins" for reading the value into the property. This allows to create scoped directory structures as you see fit. Example: `foo.bar/test.example` is less specific than `foo/bar/test.example` and so on.

You cannot use directories or files whose names start with a dot. They will be ignored, following the POSIX philosophy of hidden files and folders.

Symbolic links will be followed, so you can expose files from such hidden areas, allowing for all types of mangling with names etc. Don't link to directories, as the file monitors rely on real directories.

Kubernetes Example

You want to retrieve a (secret) value via property `foo.bar.property1`:

```
@ConfigProperty("foo.bar.property1")
```

You deployed a secret to your Kubernetes cluster:

```
apiVersion: v1
kind: Secret
metadata:
  name: foobar
type: Opaque
stringData:
  property1: "my-super-secret-value"
```

And your pod mounts it at `/home/payara/secrets/foo/bar` (only showing the relevant parts from the `Deployment` K8s YAML):

```
volumeMounts:
  - name: test-secrets
    mountPath: /home/payara/secrets/foo/bar
volumes:
  - name: test-secret
    secret:
```

```
      secretName: foobar
```

```
/ # ls -la /home/payara/secrets/foo/bar
total 3
drwxrwxrwt 3 root root  120 Nov 25 10:51 .
drwxr-xr-x 3 root root 4096 Nov 25 10:51 ..
drwxr-xr-x 2 root root   80 Nov 25 10:51 ..2020_11_25_10_51_55.283009570
lrwxrwxrwx 1 root root   31 Nov 25 10:51 ..data -> ..2020_11_25_10_51_55.283009570
lrwxrwxrwx 1 root root   15 Nov 25 10:51 property1 -> ..data/property1
```

The server instance will pick up the file and read its content as a value for property `foo.bar.property1`

JDBC Config Source

The JDBC config source reads configuration properties from a database table.

Configuration

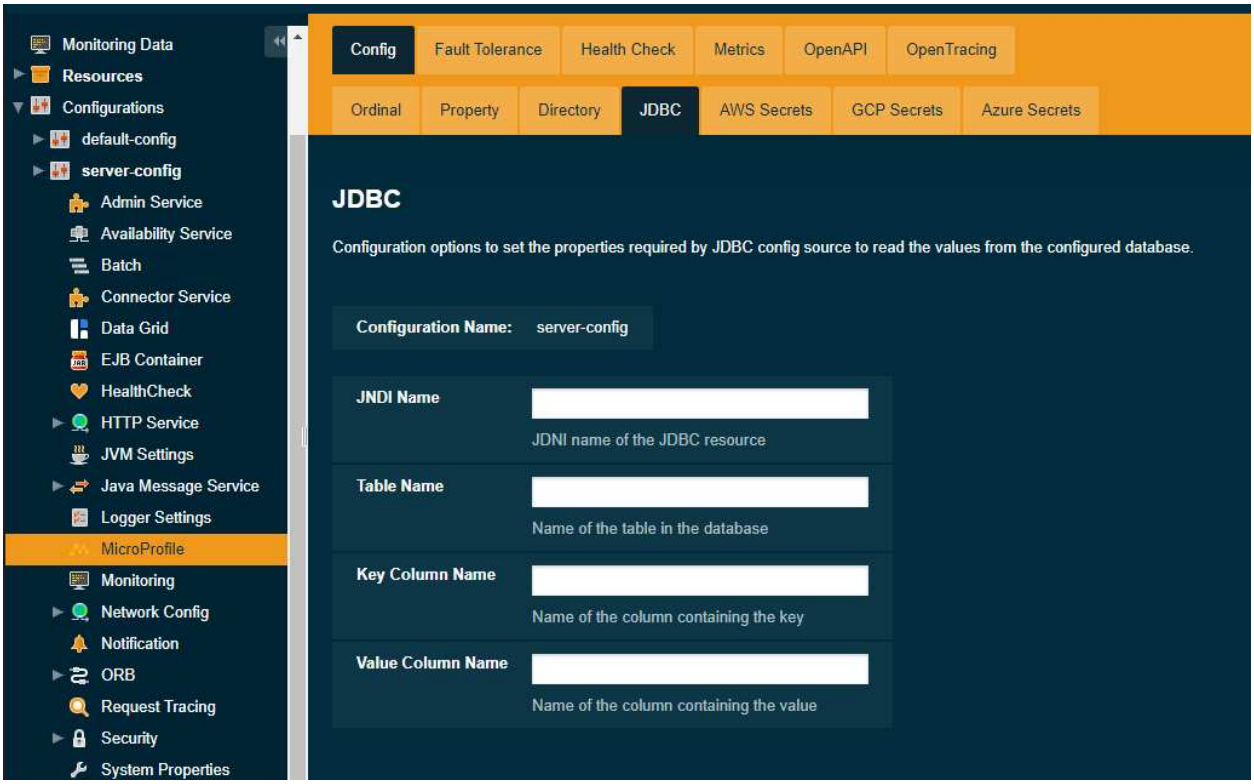
You can configure JDBC Config Source either via the admin console or the *asadmin* utility. You'll need the JNDI name of the database JDBC resources, name of a database table, name of the key column containing the key and name of the column containing the value.

The values from the key column will be used as the config property names and the values from the value column will be used as the config property value.

IMPORTANT	The type of the database columns for the *key* and *value* columns needs to be of type `*String*` or compatible.

From the Admin Console

To configure the config source from the admin console, go to `Configs` ▸ `your-config` ▸ `MicroProfile` ▸ `Config` ▸ `JDBC`.



From here you configure the JNDI name of the database JDBC resources, name of the database table, name of the key column containing the key and name of the column containing the value.

From the Command Line

To configure the JDBC config source from the command line, use the following commands:

- set-jdbc-config-source-configuration.
- get-jdbc-config-source-configuration.

Usage

Once all required options are configured. You should be able to read configuration properties from a database table.

For example, let's look at the following diagram. It shows a table named Books with columns ISBN, Author and Title.

Books		
ISBN	Authour	Title
9781973907985	Harper Lee	To Kill a Mockingbird
9780030779008	Charles Dickens	Great Expectations
9780345916419	Philip Pullman	The Subtle Knife
9780060120375	Aldous Huxley	Brave New World

If the key column name is set to `ISBN` and the value column name to `Title`. You can get the value of the Title column in relation to the ISBN column using the following command along with the ISBN value:

```
asadmin get-config-property --source jdbc --propertyName 9780030779008
Great Expectations
Command get-config-property executed successfully.
```

NOTE

Currently, the JDBC config source only supports the `get-config-property` command and doesn't support the `set-config-property` and `delete-config-property` commands.

LDAP Config Source

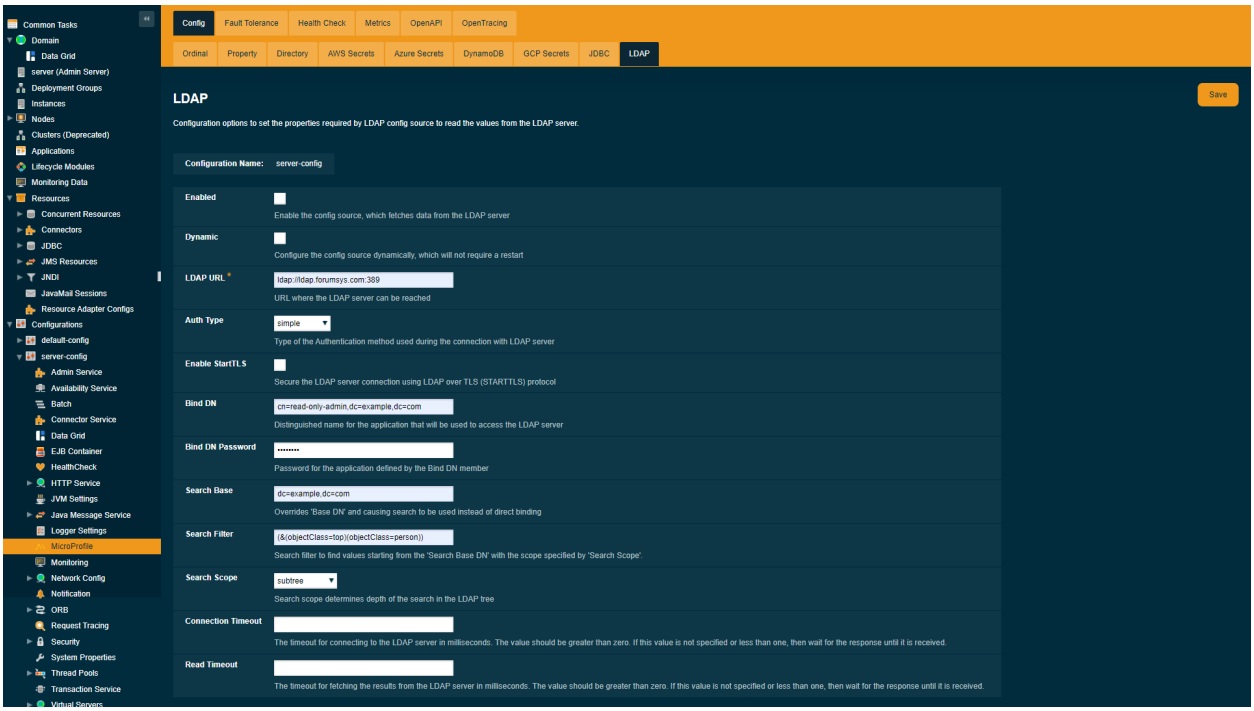
The LDAP configuration source reads properties from a predefined LDAP directory.

Configuration

The LDAP Config Source can be configured by using the Admin Console or `asadmin` commands.

From the Admin Console

To configure the config source from the admin console, go to `Configs` ▯ `your-config` ▯ `MicroProfile` ▯ `Config` ▯ `LDAP`.



Using the Asadmin Commands

The following administration commands can be used to configure the MicroProfile LDAP config source:

- set-ldap-config-source-configuration.
- get-ldap-config-source-configuration.

Usage

Once all required options are configured, you should be able to read configuration properties from the LDAP directory.

```
asadmin> get-config-property --source ldap --propertyName sn
Read Only Admin
Command get-config-property executed successfully.
```

NOTE

Currently, the LDAP config source only supports the `get-config-property` command and doesn't support the `set-config-property` and `delete-config-property` commands.

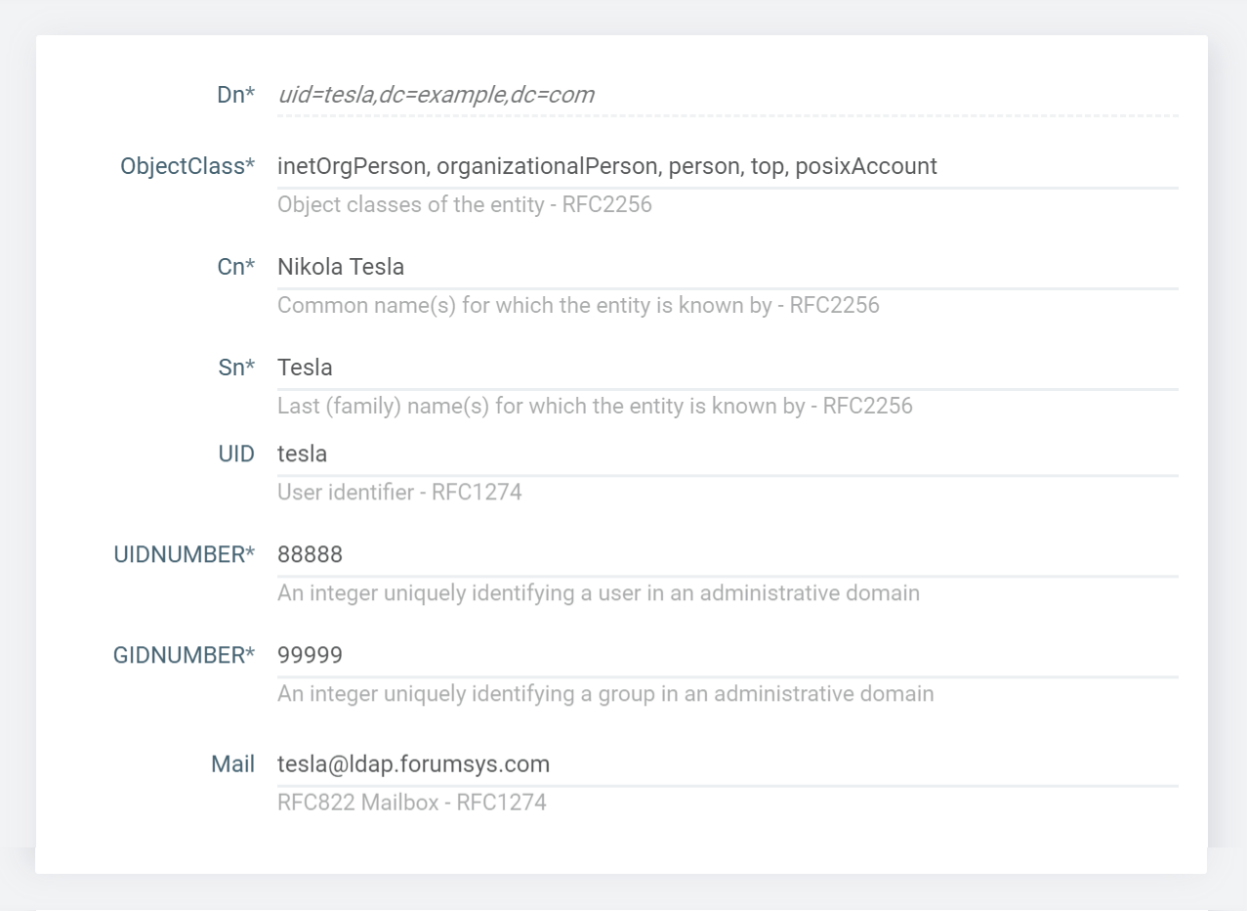
Finding User Specific Properties

If search filter and search base are not specified then properties fetched from the direct binding of `bindDn`.

```
asadmin set-ldap-config-source-configuration --enabled=true --dynamic=true --url
=ldap://ldap.forumsys.com:389 --authType=simple --bindDnPassword=password --bindDn
=cn=read-only-admin,dc=example,dc=com
```

```
asadmin get-config-property --source ldap --propertyName sn
Read Only Admin
Command get-config-property executed successfully.
```

Alternatively to fetch the user specific properties, search filter can be specified with attributes unique to specific user. For example, in the following configuration `uid` is used as user filter in the search query.



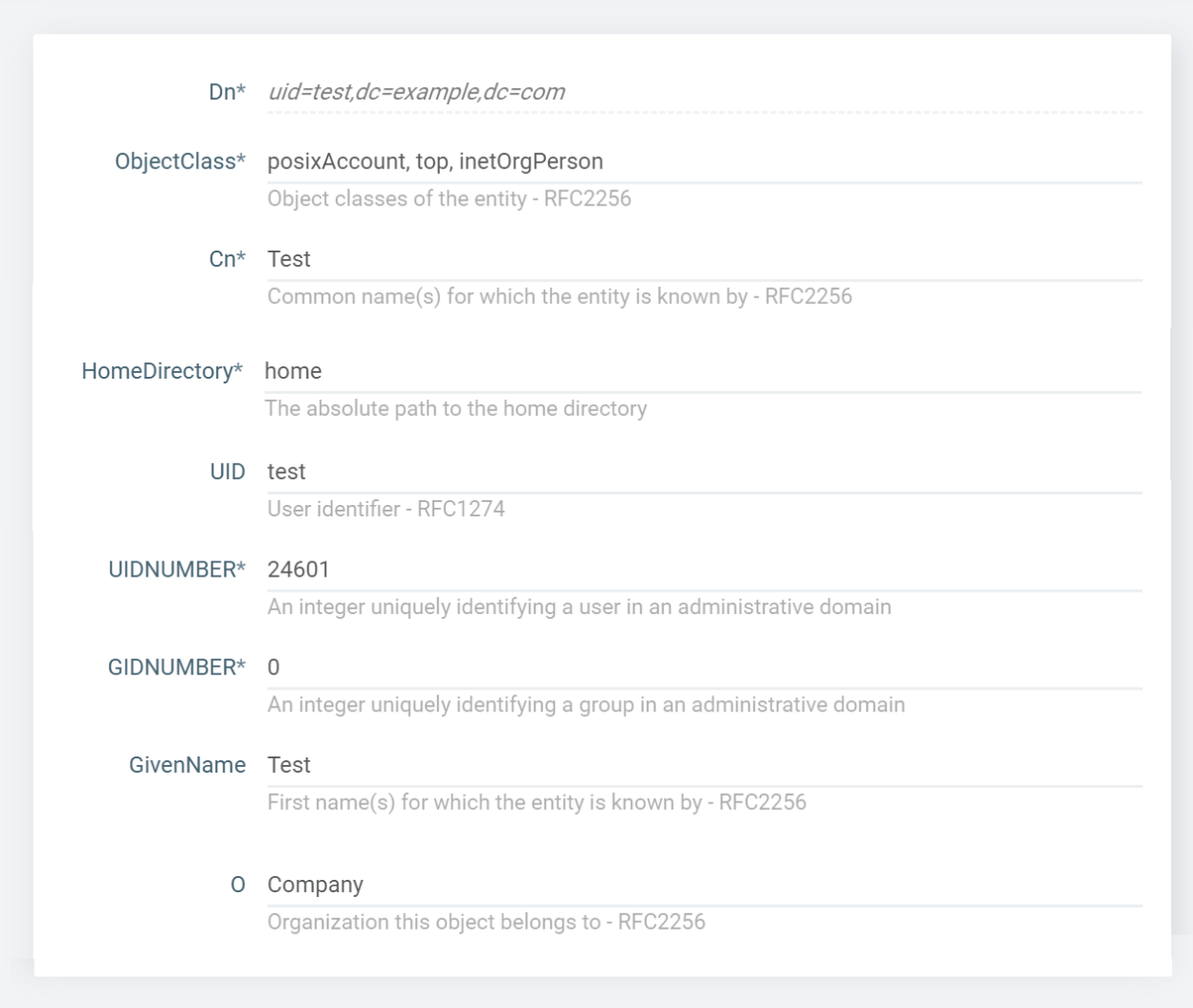

```
asadmin set-ldap-config-source-configuration --enabled=true --dynamic=true --url
=ldap://ldap.forumsys.com:389 --authType=simple --bindDNPassword=password --bindDN
=cn=read-only-admin,dc=example,dc=com --searchBase=dc=example,dc=com --searchScope
```

```
=subtree --searchFilter=(&(uid=tesla))
```

```
asadmin get-config-property --source ldap --propertyName UIDNUMBER
88888
Command get-config-property executed successfully.

asadmin get-config-property --source ldap --propertyName GIDNUMBER
99999
Command get-config-property executed successfully.

asadmin get-config-property --source ldap --propertyName MAIL
tesla@ldap.forumsys.com
Command get-config-property executed successfully.

asadmin get-config-property --source ldap --propertyName sn
Tesla
Command get-config-property executed successfully.
```

Finding Group Specific Properties

To find properties from the group of users search filter is mandatory to specify. For e.g, In the following configuration `objectClass` is used as group filter.




```
asadmin set-ldap-config-source-configuration --enabled=true --dynamic=true --url
=ldap://ldap.forumsys.com:389 --authType=simple --bindDNPassword=password --bindDN
=cn=read-only-admin,dc=example,dc=com
 --searchBase=dc=example,dc=com --searchScope=subtree --searchFilter=(&(
objectClass=posixAccount)(objectClass=organizationalPerson))
```

```
asadmin get-config-property --source ldap --propertyName sn
Tesla,Test
Command get-config-property executed successfully.
```

TOML Config Source

The TOML Config Source takes configuration properties from a TOML configuration file.

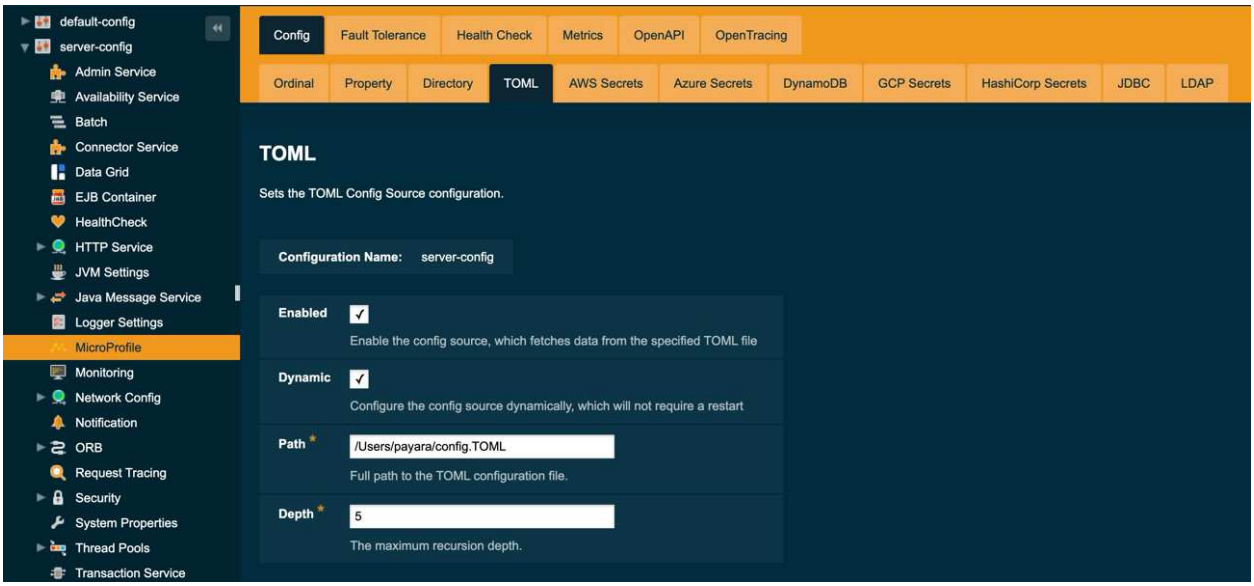
Configuration in Payara Server

You can configure TOML Config Source either via the admin console or the *asadmin* utility.

You'll need the path of the TOML configuration file and the maximum recursion depth.

From the Admin Console

To configure the config source from the admin console, go to `Configs` → `your-config` → `MicroProfile` → `Config` → `TOML`.



From here you can enter the path to the TOML configuration file and set the maximum recursion depth.

You can also decide whether to apply these changes dynamically or on the next server restart. If the config source is enabled or disabled dynamically it will take effect across the server immediately.

From the Command Line

To configure the TOML Config Source from the command line, use these commands:

- set-toml-config-source-configuration
- get-toml-config-source-configuration

Usage

Provided that all of the above sections are configured correctly, the properties can be injected into any applicable MicroProfile Config injection point as with any other Config Source. They can also be fetched using the asadmin utility.

To prevent `StackOverFlowException` issues, the `depth` parameter limits the recursion depth when reading properties from the TOML file. If the maximum recursion depth is reached, further properties will not be read.

Examples

Example TOML file

```
title = "TOML Example"

[owner]
name = "Tom Preston-Werner"
dob = 1979-05-27T07:32:00-08:00

[database]
enabled = true
ports = [ 8000, 8001, 8002 ]
```

Retrieving properties programmatically

```
Config config = ConfigProvider.getConfig();
String ownerName = config.getValue("owner.name", String.class);
```

Inject configuration properties

```
@Inject
@ConfigProperty(name="database.ports[0]")
private Optional<Integer> databasePort;
```

See Also

- TOML Documentation

Eclipse Microprofile Fault Tolerance API

Azul Payara 7.2.0 provides MicroProfile Fault Tolerance 4.0

Background

The Fault Tolerance API was created to help separate execution logic from execution. The execution can be configured with a number of fault tolerance policies.

The complete specification can be found on the Eclipse MicroProfile website.

Breaking changes introduced in MicroProfile Fault Tolerance 4.0 are listed in the official

specification under Incompatible Changes

Additions to the Fault Tolerance Spec

In addition to the functionality detailed in the specification document, Azul Payara has implemented additional features like:

Configurable Executor Services

On Payara Server, the managed and scheduled managed executor pool used to generate threads for the service can be configured via the `set-fault-tolerance-configuration` Asadmin command.

The executor service is used to execute methods annotated with `@Asynchronous`, whereas the scheduled executor service is used by the `CircuitBreaker` and `Timeout` interceptors for their timeout operations (scheduling the circuit breaker to be set to half open, and just timing out respectively).

Alternative @Asynchronous Annotations

The Azul Payara configuration property `MP_Fault_Tolerance_Alternative_Asynchronous_Annotations` can be used to specify a comma separated list of fully qualified class names of those annotations that should have the same effect as the `@Asynchronous` defined in the Fault Tolerance specification.

These annotations do not have to be interceptor bindings.

For example:

```
MP_Fault_Tolerance_Alternative_Asynchronous_Annotations=jakarta.ejb.Asynchronous
```

Annotation Priority

As per the Fault Tolerance specification, interactions between annotations are handled effectively nesting computation in the following way (skipping handling for annotations not present):

1. `Asynchronous`
2. `Fallback`
3. `Retry`
4. `CircuitBreaker`
5. `Timeout`
6. `Bulkhead`
7. (calling annotated method; might be wrapped by other interceptors)

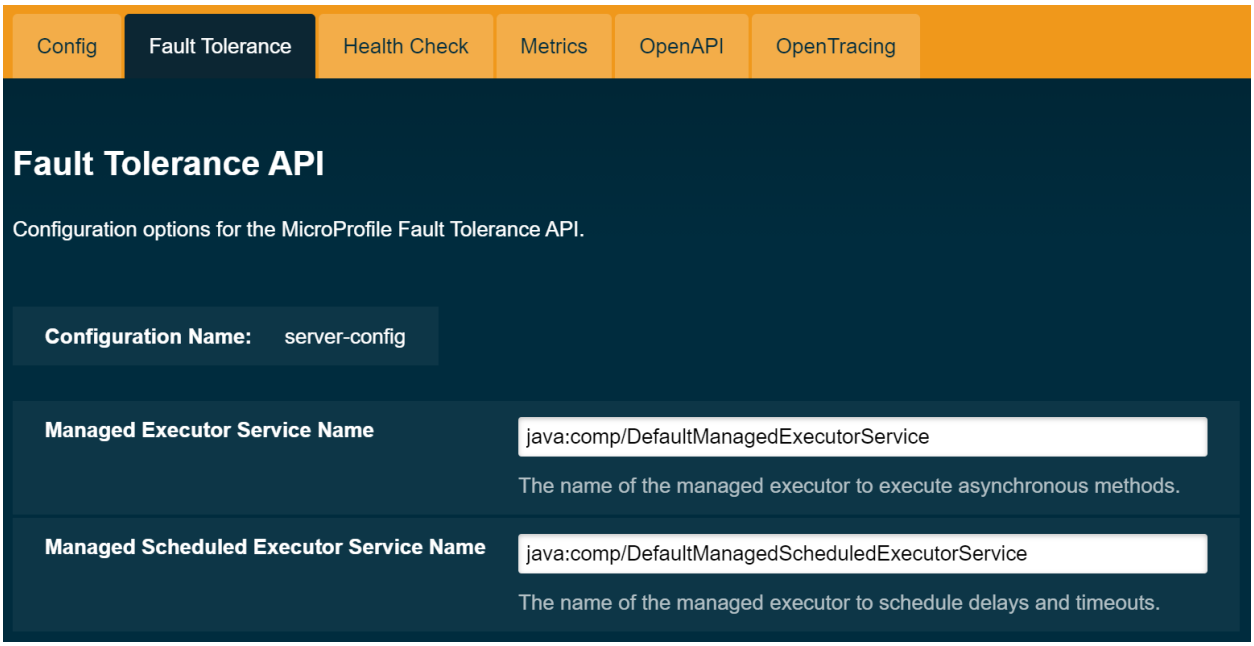
As specified the interceptor priority can be changed using the property `mp.fault.tolerance.interceptor.priority` affecting all annotations including alternative ones.

Fault Tolerance Configuration

On Payara Server the Fault Tolerance service can be configured by using the Admin Console or Asadmin CLI commands.

Using the Admin Console

To configure the Fault Tolerance in the Admin Console, go to Configuration [instance-configuration (like `server-config`)] MicroProfile Fault Tolerance:



Using Asadmin Commands

Use the following commands to modify the configuration settings of the MicroProfile Fault Tolerance service:

- set-fault-tolerance-configuration
- get-fault-tolerance-configuration

Upgrading from Eclipse Microprofile 3.x to 4.x

MicroProfile 4.0 brings with it a number of changes to MicroProfile Fault Tolerance. There are two incompatible changes with the previous version of MicroProfile Fault Tolerance.

Metric Names and Scopes Changed

The metrics added automatically by MicroProfile Fault Tolerance have been updated to take advantage of support for metric tags which were added to MicroProfile Metrics in version 2.0. As a result, some information which was previously contained in the metric name is now instead included in tags.

In addition, metrics have moved from the `application` scope to the `base` scope for consistency with other MicroProfile specifications. Note that this means:

- Metrics are now exported under `/metrics` and `/metrics/base`, instead of `/metrics` and `/metrics/application` as in previous versions.
- In the JSON format, when metrics are retrieved from `/metrics` they appear in the base object rather than the application object.
- In the OpenMetrics format, the names are prefixed with `base_` instead of `application_`.

There unfortunately is not a workaround for this change, users must simply ensure that any dashboard or application making use of these metrics must be updated to look at the new locations.

Example

Old format:

```
application:ft.<name>.timeout.callsTimedOut.total
```

New format:

```
base:ft.timeout.calls.total{method="<name>", timedOut="true"}
```

Lifecycle of Circuitbreakers and Bulkheads Specified

In previous versions of MicroProfile Fault Tolerance, the lifecycle of `CircuitBreaker` and `Bulkhead` annotations was not specified. These fault tolerance strategies hold state between invocations, so their lifecycle is important for correct functioning.

`CircuitBreaker` and `Bulkhead` are now treated as singletons identified by the annotated bean class and the guarded method. This means that if a `RequestScoped` bean has a method annotated with `CircuitBreaker` and/or `Bulkhead`, all invocations of that method will share the same `CircuitBreaker` and/or `Bulkhead` state, even though each request will have a different instance of the annotated bean.

There is unfortunately no workaround for this change in behaviour.

Eclipse Microprofile Health Check API

Azul Payara 7.2.0 provides MicroProfile HealthCheck 4.0

Background

The Health Check API was created to allow easier probing of the state of computing nodes that live in a cloud infrastructure environment.

The complete specification can be found on the Eclipse MicroProfile website.

Breaking changes introduced in MicroProfile Health 4.0 are listed in the official specification under Incompatible Changes

REST Endpoints

The MicroProfile HealthCheck API defines the following list of REST endpoints that are available from the root context at `/health/${type}` by default:

Context	Type	Description
`/health/live`	Liveness	On invocation, all registered health checks annotated with `@Liveness` are performed and the results aggregated.
`/health/ready`	Readiness	On invocation, all registered health checks annotated with `@Readiness` are performed and the results aggregated.
`/health/started`	Startup	On invocation, all registered health checks annotated with '@Startup' are performed and the results aggregated.
`/health`	Readiness + Liveness + Startup	On invocation, all registered health checks annotated with `@Liveness`, `@Readiness`, or `@Startup` are performed and the results aggregated.

TIP

On Payara Server, this root endpoint can be altered using the `set-microprofile-healthcheck-configuration` asadmin command to prevent collisions with other HTTP endpoints that might need to use the `health` context root.

IMPORTANT

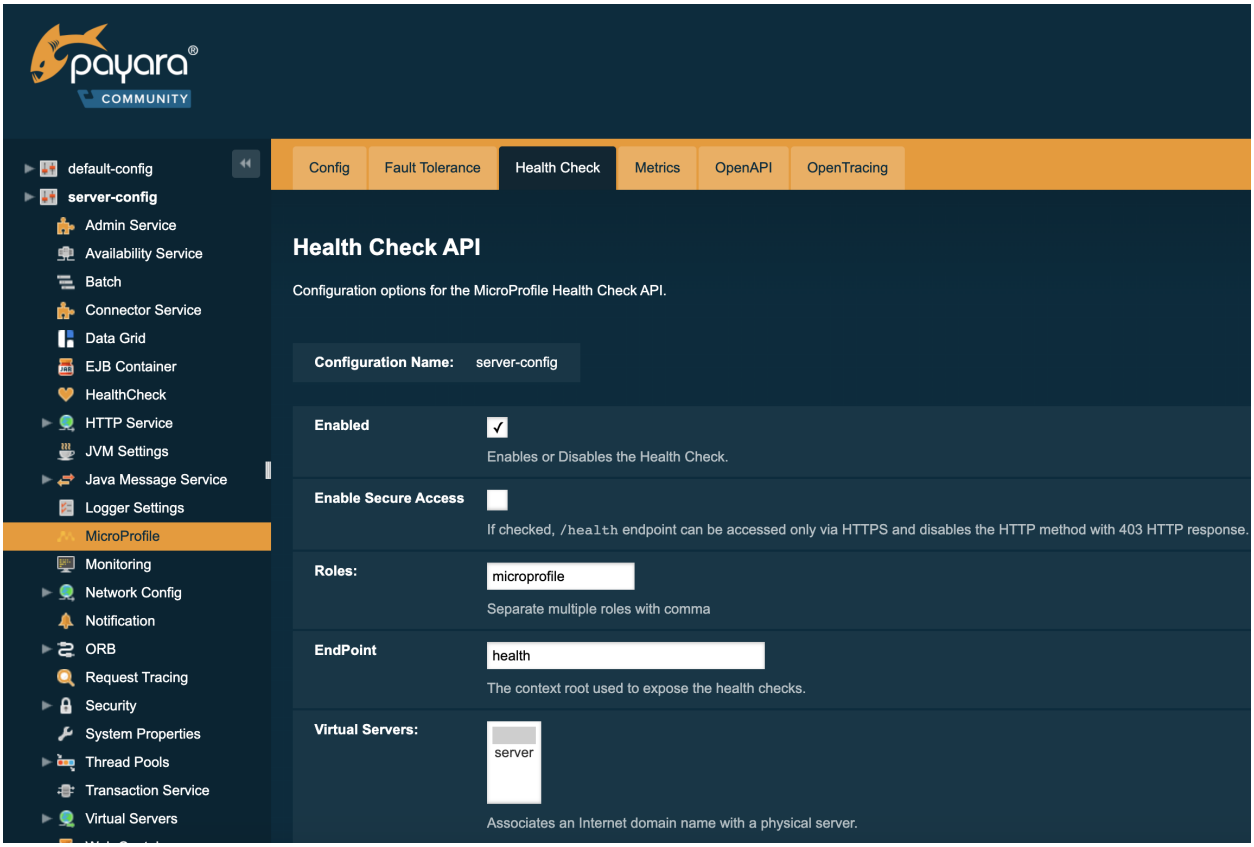
This endpoint will not be available until at least one application has been deployed.

Health Check Configuration

On Payara Server, the MicroProfile HealthCheck settings can be configured by using Admin Console or Asadmin commands.

Using the Admin Console

To configure the Health Check in the Admin Console, go to Configuration [instance-configuration (like `server-config`)] MicroProfile Health Check:



Using Asadmin Commands

The following administration commands can be used to configure the MicroProfile Healthcheck service:

- set-microprofile-healthcheck-configuration
- get-microprofile-healthcheck-configuration

IMPORTANT	Changes made with this command will only be applied when the server or instance is restarted.

Security Configuration

By default, HealthCheck endpoints bind to the root context application which is the `default-web-module` (also known as **`docroot`**) system application and `thedefault-web-module` application, which is secured under the default realm (`file`) of the server.

If a user application is deployed in the empty context-root, then the security

configuration of this application will be shared by the HealthCheck endpoints, so *exert extreme caution* when making these changes.

Upgrading from Eclipse Microprofile 3.x to 4.x

MicroProfile 4.0 brings with it a number of changes to MicroProfile Health. The main incompatible change brought in by this upgrade is the removal of the deprecated `@Health` annotation.

There isn't a workaround for this, the annotation is simply gone: all health checks must be registered as a `@Readiness`, `@Liveness`, or `@Startup` check.

Eclipse Microprofile JWT Authentication API

Azul Payara 7.2.0 provides MicroProfile JWT Authentication 2.1.

Background

The JWT Authentication API was designed to provide application callers with the ability to authenticate themselves using a JWT token. A JWT token is essentially a string of JSON with fields for specifying the caller/username and the groups the caller is in.

To prevent tampering, the JSON token is cryptographically signed. Note that it's only signed, not fully encrypted. This means that the caller/username and the groups are in plain text, but because of the signature can't be changed.

The complete specification can be found on the Eclipse MicroProfile website.

Breaking changes introduced in MicroProfile JWT Authentication 2.1 are listed in the official specification under Incompatible Changes

IMPORTANT	The MicroProfile JWT Authentication specification currently does not define any means for preemptive authentication, that is, authenticating when the target resource (such as a JAX-RS end-point) is not protected / constrained by a security role. Practically this means that if a JWT token is sent alongside a request to a public / unchecked resource, MicroProfile JWT

> implementations will ignore it. This limitation will be addressed in a later version of the JWT specification.

Jakarta Security and JWT

Payara Server and Payara Micro implement the JWT authentication mechanism as a normal Jakarta Security (JSR 375) authentication mechanism.

This specifically means that when MP JWT authentication is used on Payara Server or Payara Micro, there's an `HttpAuthenticationMechanism` CDI bean enabled that can be intercepted or decorated using the standard CDI APIs. Likewise, there's an `IdentityStore` CDI bean enabled, which can be intercepted or decorated too.

By providing additional `IdentityStore` beans, an application can, if needed, augment the JWT identity store, for instance by providing extra roles.

Vending Tokens

A JWT token, like most API tokens, is typically vended from a website for use with the services provided by that site. For instance, from Google or Facebook can obtain such a token for usage with their respective APIs.

Though JWT tokens are more universal in format, they are still intended for use with a specific service (website / API). This is encoded in the **mandatory** `iss` ("issuer") field. The value of the field can be anything that the intended service can recognize (see below).

For example:

```
{
    "iss": "fish.payara.example",
    "jti": "a-123",
    "sub": "24400320",
    "aud": "s6BhdRkqt3",
    "upn": "test",
    "groups": [
        "architect",
        "master",
        "leader",
        "dev"
    ]
```

```
}
```

A JWT also **must** be signed, and in the case of MicroProfile JWT this **must** be `RSASSA-PKCS-v1_5` using the SHA-256 hash algorithm.

One way to do this signing is by first generating an SSH keypair as follows:

1. Open a terminal
2. Generate the base key by entering: `openssl genrsa -out baseKey.pem`
3. From the base key generate the PKCS#8 private key: `openssl pkcs8 -topk8 -inform PEM -in baseKey.pem -out privateKey.pem -nocrypt`
4. And generate the public key: `openssl rsa -in baseKey.pem -pubout -outform PEM -out publicKey.pem`

Then put `privateKey.pem` on the root of the classpath (for instance, in a Maven project put it in `src/main/resources`) and use the following code to create the signed JSON token as a string:

```
@RequestScoped
public class TokenGenerator{

    public static String generateJWTString(String jsonResource) {
        byte[] byteBuffer = new byte[16384];
        currentThread().getContextClassLoader()
                       .getResource(jsonResource)
                       .openStream()
                       .read(byteBuffer);

        JSONParser parser = new JSONParser(DEFAULT_PERMISSIVE_MODE);
        JSONObject jwtJson = (JSONObject) parser.parse(byteBuffer);

        long currentTimeInSecs = (System.currentTimeMillis() / 1000);
        long expirationTime = currentTimeInSecs + 1000;

        jwtJson.put(Claims.iat.name(), currentTimeInSecs);
        jwtJson.put(Claims.auth_time.name(), currentTimeInSecs);
        jwtJson.put(Claims.exp.name(), expirationTime);

        SignedJWT signedJWT = new SignedJWT(new JWSHeader
                                            .Builder(RS256)
                                            .keyID("/privateKey.pem")
                                            .type(JWT)
                                            .build(), parse(jwtJson));

        signedJWT.sign(new RSASSASigner(readPrivateKey("privateKey.pem")));
```

```
        return signedJWT.serialize();
    }

    public static PrivateKey readPrivateKey(String resourceName){
        byte[] byteBuffer = new byte[16384];
        int length = currentThread().getContextClassLoader()
                                    .getResource(resourceName)
                                    .openStream()
                                    .read(byteBuffer);

        String key = new String(byteBuffer, 0, length).replaceAll("-----BEGIN
(.*)-----", "")
                                                      .replaceAll("-----END
(.*)----", "")
                                                      .replaceAll("\r\n", "")
                                                      .replaceAll("\n", "")
                                                      .trim();

        return KeyFactory.getInstance("RSA")
                         .generatePrivate(new PKCS8EncodedKeySpec(Base64.
getDecoder().decode(key)));
    }
}
```

The code here assumes the location of the raw JSON file on the classpath being passed in, the private key on the root of the classpath and the result as a string.

If other input/outputs are needed that should be easy to do using the above code as a starting point.

Accepting Tokens

Public Key Configuration

In order to validate the signature of a provided token is valid, an application that uses JWT Authentication has to provide the public key to the MicroProfile JWT Authentication implementation either using the standard MicroProfile Configuration options or using the vendor-specific option.

`mp.jwt.verify.publickey`

The `mp.jwt.verify.publickey` microprofile config property allows the Public Key text itself to be supplied as a string.

For e.g :

`mp.jwt.verify.publickey=joer4fghieEM3UmZQcFRvNzM2fhMnJ6QV45ghRCdTQ1SnYwdXBkRVpjc54645jNJc65XltamJaUmtwZ1RSOEIxOWJfcl`

`mp.jwt.verify.publickey.location`

The `mp.jwt.verify.publickey.location` microprofile config property allows for an external or internal location of Public Key to be specified. The value may be a relative path or a URL.

For e.g : `mp.jwt.verify.publickey=/META-INF/publicKey.pem`

Public Key Vendor-Specific Configuration

In Payara this is done by placing the public key such as generated above as `publicKey.pem` on the root of the application's classpath. For example, when using a Maven project as `src/main/resources/publicKey.pem`.

NOTE	Vendor-specific option for supplying the public key will always take precedence.

Issuer Configuration

Next to providing the public key, an application that uses JWT Authentication has to provide the issuer (corresponding to the `iss` field in the JSON token) it's willing to accept.

Issuer can be provided to the MicroProfile JWT Authentication implementation either using the standard MicroProfile Config option or using the vendor-specific option.

`mp.jwt.verify.issuer`

The `mp.jwt.verify.issuer` microprofile config property allows for the expected value of the `iss` claim to be specified.

Issuer Vendor-Specific Configuration

For Azul Payara, this is done by placing a properties file named `payara-mp-jwt.properties` on the root of the application's classpath. For example, when using a Maven project as `src/main/resources/payara-mp-jwt.properties`. The properties file should contain the key `accepted.issuer` with as value the same value that of the `iss` field in the vended token, e.g. `fish.payara.example` as per the example JSON token shown above.

NOTE	Vendor-specific option for supplying the issuer will always take precedence.

Namespaced Claims Configuration

Authentication services (like auth0, connect2id) offer the chance to add custom claims to JWT tokens but also enforce a namespaced format to avoid possible collisions with standard OpenID Connect claims.

In Azul Payara, namespaced claims configuration is done by placing a properties file named `payara-mp-jwt.properties` on the root of the application's classpath.

For example, when using a Maven project as `src/main/resources/payara-mp-jwt.properties`. These properties file should contain the Boolean property `enable.namespace` and the optional property `custom.namespace`.

enable.namespace

If this is true, the default `https://payara.fish/mp-jwt/` namespace will be used and the parser will look out for namespaced claims.

For example:

If following JSON is the token payload and `enable.namespace` property is `true`:

```
{
  "https://payara.fish/mp-jwt/groups": ["admin", "read", "write"],
  "https://payara.fish/mp-jwt/upn": "test",
  "iss": "https://test.auth.com/",
  "sub": "5b2856bf8763ef356976dca3"
}
```

Then the JSON Parser search for namespace prefixed claims, remove the namespace from claim name, allow the processing of the token as usual.

custom.namespace

When the `custom.namespace` property is set, it will always take precedence over the default namespace and be used instead.

Disabling Type Claim Verification

The MicroProfile JWT Authentication specification currently mandates that the type claim (`typ`) of any authorization token parsed by the container is present and is set to the `JWT` value.

However, the current RFC document (RFC 7519) that defines the JWT standard states that this claim is **optional**:

> 5.1. "typ" (Type) Header Parameter
>
> The "typ" (type) Header Parameter defined by [JWS] and [JWE] is used by JWT applications to declare the media type [IANA.MediaTypes] of this complete JWT. This is intended for use by the JWT application when values that are not JWTs could also be present in an application data structure that can contain a JWT object; the application can use this value to disambiguate among the different kinds of objects that might be present.
>
> ...
>
> Use of this Header Parameter is OPTIONAL.

For this reason, some third-party token issuers may generate tokens that are not compatible with the MicroProfile JWT specification. Azul Payara allows to set this verification off, so you can use the `disable.type.verification` custom property and set its value to `true` to this effect.

NOTE	This property has to be defined in the `payara-mp-jwt.properties` configuration file described in the previous section.
IMPORTANT	Keep in mind that tokens which are missing their type claim and are propagated to other services running on other Eclipse MicroProfile runtimes might be rejected, as the specification mandates the inclusion of the claim.

Caching the Public Key

By default, the public key retrieved by the `mp.jwt.verify.publickey.location` configuration property will be cached in memory for 5 minutes after being read from either a local file or a remote location. You can modify this *"time-to-live"* which determines how long the key stays cached in memory in the case you are dealing with long-lived keys, this is done by setting the `publicKey.cache.ttl` custom property.

NOTE This property has to be defined in the `payara-mp-jwt.properties` configuration file described in the previous section.

NOTE If the public key is retrieved from a remote HTTP(S) location, and the server's response includes a `Cache-Control: max-age` header with a valid positive value, Payara will use that value as the cache duration instead of the `publicKey.cache.ttl` setting

IMPORTANT The value of the `publicKey.cache.ttl` property is defined in milliseconds, so keep this in mind when modifying the property

Retaining the Cached Public Key on Retrieval Errors

When a public key is retrieved from a remote location as specified by the `mp.jwt.verify.publickey.location` configuration property, Payara caches it in memory for the duration defined by the `publicKey.cache.ttl` property (default 5 minutes). Once this time-to-live (TTL) period expires, the runtime will attempt to re-fetch the key from the remote location.

In certain environments, however, the remote public key location may become temporarily unavailable due to network interruptions or service downtime. In such cases, the cached key would normally be discarded after its TTL expires, potentially causing JWT verification failures even if the cached key itself remains valid.

To improve resilience in these scenarios, you can configure Payara to **retain the cached key for an additional period** if the runtime cannot reach the remote location when attempting to refresh it. This behavior is controlled by the

`publicKey.cache.retain-on-error-duration` custom property.

When this property is set, once the cached key expires and the system fails to fetch the updated key, the previously cached key will remain valid and continue to be used for the duration specified by this property. If the remote location becomes available again within this period, the cached key will be refreshed as normal. If the retain-on-error period also expires without a successful refresh, the cache will be invalidated and JWT verification will fail until the key can be fetched again.

NOTE	This property must be defined in the `payara-mp-jwt.properties` configuration file described in the previous section.
IMPORTANT	The value of the `publicKey.cache.retain-on-error-duration` property is defined in **milliseconds**. For example, a value of `3600000` will retain the cached key for an additional 1 hour in the event of a remote key retrieval failure.

Activating JWT Authentication

An application activates the JWT authentication mechanism and identity store by annotating a class in the application, for instance, the JAX-RS `Application` class, with `@LoginConfig(authMethod = "MP-JWT")`.

Protecting Jakarta REST Services Endpoints

MicroProfile JWT Authentication specifies that Jakarta REST services endpoints are to be secured by using the `jakarta.annotation.security.RolesAllowed` annotation. Note that while this is a general annotation, for Jakarta EE it's only EJBs that interprets this.

The Jakarta REST Services specification itself does not specify that this annotation should work on resource classes or methods and hence most implementations do not support it out of the box. Azul Payara supports this behaviour out of the box in a proprietary manner. This support holds for all types of authentication mechanisms, e.g. `BASIC`, and not just `JWT`.

For example:

```
@ApplicationScoped
@Path("/resource")
@Produces(TEXT_PLAIN)
public class Resource {

    @Inject
    private Principal principal;

    @GET
    @Path("/protected")
    @RolesAllowed("architect")
    public String protectedResource() {
        return
            "This is a protected resource \n" +
            "web username: " + principal.getName() + "\n";
    }
}
```

Accessing a Protected Endpoint

With the `generateJWTString()` method as presented above and the JWT token residing in a file called `jwt-token.json` on the classpath, a request to a JWT protected endpoint can be done as follows using the Jakarta REST Services client API:

```
public class Application{

    public static void main(String[] args){
        var response = newClient()
                        .target(
                            URI.create(new URL(base, "resource/protected"
).toExternalForm()))
                        .request(TEXT_PLAIN)
                        .header(AUTHORIZATION, "Bearer " + generateJWTString("jwt-
token.json"))
                        .get(String.class);
    }
}
```

With `base` being the context where the application is deployed, e.g. `http://example.com/myapp`

Switching off @Rolesallowed Support

The out-of-the-box support of `@RolesAllowed` for Jakarta REST Services resources can be switched off by setting the `<jaxrs-roles-allowed-enabled>` tag in `WEB-INF/payara-web.xml` to `false`.

For more information see the documentation for the jaxrs-roles-allowed-enabled element.

Metrics

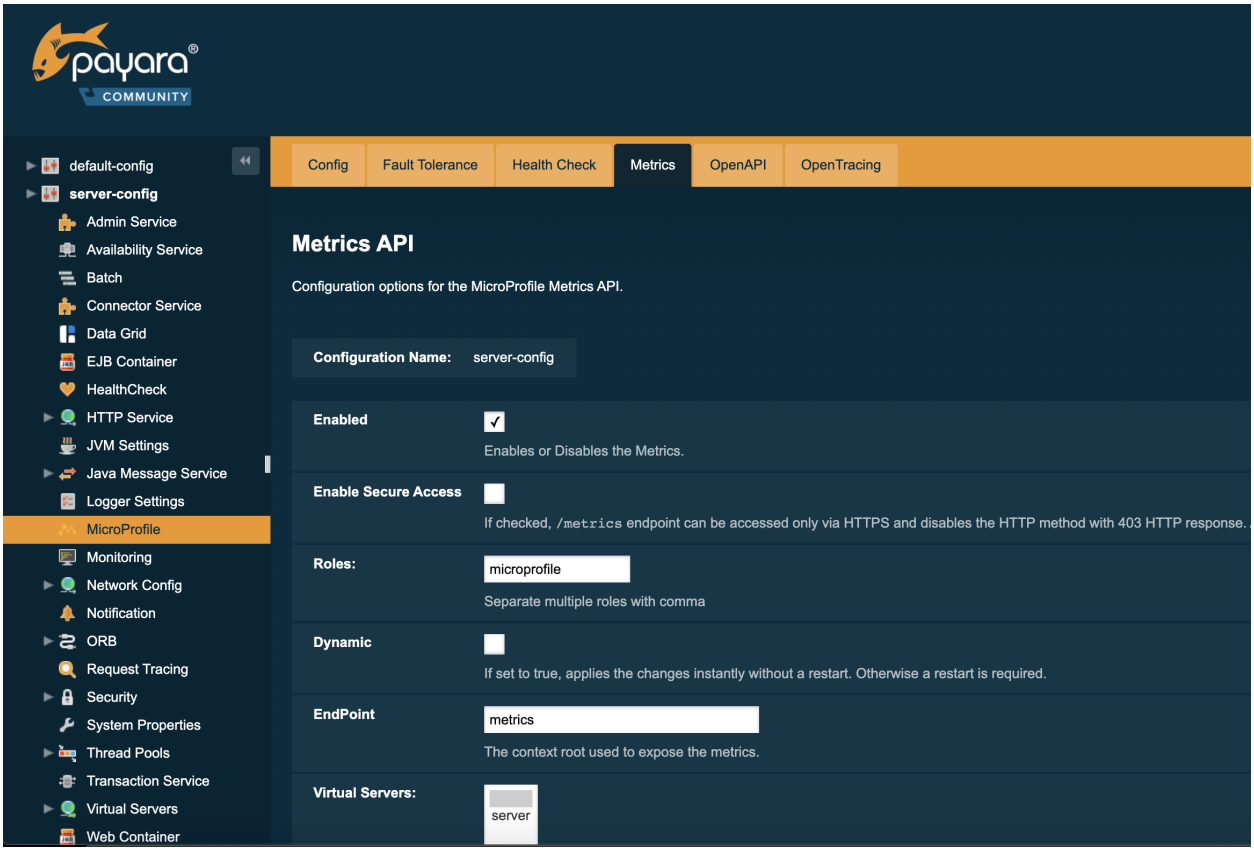
Metrics Configuration in Azul Payara

On Payara Server, the MicroProfile Metrics service can be configured by using Admin Console or Asadmin commands.

Payara Micro doesn't have a convenient solution for its MicroProfile Metrics service configuration yet.

Using the Admin Console

To configure the Metrics in the Admin Console, go to Configuration [instance-configuration (like `server-config`)] MicroProfile Metrics:



Using Asadmin Commands

Use the following commands to manipulate the configuration settings of the service:

- set-metrics-configuration
- get-metrics-configuration

Security Configuration Considerations

By default, the MicroProfile Metrics endpoint binds to the root context application which is the `default-web-module (also known as docroot) system application and thedefault-web-module` application, which is secured under the default realm (`file`) of the server domain.

If a user application is deployed in the empty **context-root**, then the security configuration of this application will be shared by the MicroProfile Metrics endpoint, so *exert extreme caution* when making these changes.

REST Endpoint

MicroProfile Metrics specifies that all metrics are published at the endpoint `/metrics`. For both Payara Server and Payara Micro, this means that the endpoint can be found on the HTTP port (default `8080`). By visiting these endpoints in a browser, the Prometheus text format can be viewed directly.

Metric Scopes

The Metrics specification defines four *Scopes*, **Base**, **Vendor**, **Application** and **Custom** which correspond to the following purposes:

- **Base**: Base metrics are a set of optional metrics that a compliant server may provide. A server that provides this kind of metric can implement all, some of them or none of them. A reason to choose a different set of metrics is that the metrics libraries they are using in their implementation come with metrics with a similar purpose to the spec-described base metrics.

  These metrics can be accessed under `/metrics?scope=base`.

  These metrics must be tagged with `mp_scope=base`.

- **Vendor**: Vendor metrics are specific to each implementation and are intended to

offer metrics which are unique to each vendor.

These metrics can be accessed under `/metrics?scope=vendor`.

These metrics must be tagged with `mp_scope=vendor`.

- **Application**: Application metrics are reserved for those metrics which are to be exposed from the application by using the developer API.

  These metrics can be accessed under `/metrics?scope=application`.

  These metrics must be tagged with `mp_scope=application`.

- **Custom Scopes**: Applications may also define their own scope names by tagging metrics with a custom scope name (other than base, vendor or application).

  These metrics can be accessed under `/metrics?scope=scope_name`.

  These metrics must be tagged with `mp_scope=scope_name`.

  For example:

```
@RequestScoped
public class Component{
    @Gauge(name = "gaugeMethod", unit = MetricUnits.NONE, scope = "mygauge")
    public long getGauge() {
        log.info("Private Gauge:"+getPrivateGauge());
        return gauge;
    }
}
```

Custom Vendor Metrics

Azul Payara supports exporting the JMX Beans and HealthCheck stats to the Vendor Metrics.

JMX Mbeans

JMX MBeans can be exposed as **Custom Vendor Metrics** by the user by supplying a custom metrics.xml file in the `domain-dir/config/` directory with the following format:

```
<config>
    <base>
        ...
    </base>
    <vendor>
        <!--sample metadata-->
        <metadata>
            <tags>
                <tag>
                    <name>operatingSystem</name>
                    <value>Linux</value>
                </tag>
            </tags>
            <name>system.cpu.load</name>
            <mbean>java.lang:type=OperatingSystem/SystemCpuLoad</mbean>
            <type>gauge</type>
            <unit>none</unit>
            <displayName>System CPU Load</displayName>
            <description>Recent CPU usage for the whole system.</description>
        </metadata>
    </vendor>
</config>
```

IMPORTANT	Payara Server has a more extensive range of MBeans available than Payara Micro, but these *AMX* MBeans need to be enabled through the *Monitoring* section of the admin console before they will be usable. Each of these AMX MBeans are also lazily loaded so, for example, JDBC connection pool MBeans will not be visible until an application that uses the connection pool is deployed.

Templating Metrics for AMX Mbeans

It is possible to create a template for a metric based on AMX MBeans properties for multiple resource configurations or similar objects (like JDBC connection pools for example).

Simple Templating

You can use the `%s` placeholder to define a template in the metadata definition of the metric using the `mbean` element. The server will look for all MBeans that match the expression defined and will generate metrics for each one of them.

IMPORTANT	You will need to use the `%s` placeholder in the name element as

> well to allow each metric to have a different name and avoid collisions.

Example

Here's a simple example of defining a custom metric definition template that will generate metrics for the number of free connections available to all JDBC connections pools:

```
<metadata>
     <tags>
        <tag>
            <name>name</name>
            <value>%s</value>
        </tag>
     </tags>
     <name>jdbc.connection.pool.numconnfree</name>
     <mbean>amx:pp=/mon/server-mon[server],type=jdbc-connection-pool-
mon,name=resources/%sPool/numconnfree#current</mbean>
     <type>counter</type>
     <unit>none</unit>
     <displayName>numconnfreePool</displayName>
     <description>The total number of free connections in the pool as of the last
sampling.</description>
</metadata>
```

Which will yield the following results:

```
# TYPE vendor_jdbc_connection_pool_numconnfree_total counter
# HELP vendor_jdbc_connection_pool_numconnfree_total The total number of free
connections in the pool as of the last sampling.
vendor_jdbc_connection_pool_numconnfree_total{name="resources/DerbyPool"} 0
vendor_jdbc_connection_pool_numconnfree_total{name="resources/H2Pool"} 0
vendor_jdbc_connection_pool_numconnfree_total{name="resources/__TimerPool"} 0 0
```

Instance Names

Payara provides the placeholder `${instance}`. It is replaced with the name of the Payara Server instance, where the metrics is running:

```
<metadata>
    <name>requestcount.${instance}</name>
    <mbean>amx:type=web-request-mon,pp=/mon/server-
mon[${instance}],name=clusterjsp/server/requestcount#count</mbean>
    <type>gauge</type>
    <unit>none</unit>
    <displayName>Request count for application</displayName>
```

```
</metadata>
```

Here's the sample output of this metric being queried on a server instance named `i1`:

```
vendor_requestcount_i1 1
```

Advanced Templating

If you need more control on the attributes of specific MBeans used in a metric definition, you can use the `${key}`, `${attribute}` and `${subattribute}` placeholders to define more fine-grained metrics.

These placeholders are used as follows:

`key`: Used as a placeholder of the name (or part of the name) of an AMX MBean.

`attribute`: Used as a placeholder for the name of an **attribute** of an AMX MBean.

`subattribute`: Used as a placeholder for the name of a **sub-attribute** (or second-level attribute), which corresponds to the attribute of a `CompositeData` object.

IMPORTANT	You will need to use the `${key}`, `${attribute}` and `${subattribute}` placeholders in the `name` element as well to prevent name collisions between different templated metrics.

Example

Here's an example of defining a custom metric definition template with these placeholders that will generate a counter metric for each specific connection metric that is available to all JDBC connection pools:

```
<metadata>
     <name>jdbc.connection.pool.${key}Pool.${attribute}#${subattribute}</name>
     <mbean>amx:pp=/mon/server-mon[server],type=jdbc-connection-pool-
mon,name=resources/${key}Pool/${attribute}#${subattribute}</mbean>
     <type>counter</type>
     <unit>none</unit>
</metadata>
```

Which will yield the following results:

```
# TYPE
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconnsuccessfullymatched#s
tart_time counter
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconnsuccessfullymatched#s
tart_time 1540463722554
# TYPE vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconncreated#count
counter
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconncreated#count 0
# TYPE
vendor:jdbc_connection_pool_resources/_derby_pool_pool_connrequestwaittime#last_sam
ple_time counter
vendor:jdbc_connection_pool_resources/_derby_pool_pool_connrequestwaittime#last_sam
ple_time -1
# TYPE
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconnused#start_time
counter
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconnused#start_time
1540463106138
# TYPE
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconnused#last_sample_time
counter
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconnused#last_sample_time
1540463722554
# TYPE
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconntimedout#start_time
counter
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconntimedout#start_time
1540463722554
# TYPE
vendor:jdbc_connection_pool_resources/_derby_pool_pool_connrequestwaittime#start_ti
me counter
vendor:jdbc_connection_pool_resources/_derby_pool_pool_connrequestwaittime#start_ti
me 1540463722554
# TYPE
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconnfree#start_time
counter
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconnfree#start_time
1540463106138
# TYPE
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconnfailedvalidation#coun
t counter
vendor:jdbc_connection_pool_resources/_derby_pool_pool_numconnfailedvalidation#coun
t 0
......
# TYPE
vendor:jdbc_connection_pool_resources/_h2_pool_pool_numconnsuccessfullymatched#star
t_time counter
vendor:jdbc_connection_pool_resources/_h2_pool_pool_numconnsuccessfullymatched#star
t_time 1540463722554
# TYPE vendor:jdbc_connection_pool_resources/_h2_pool_pool_numconncreated#count
counter
vendor:jdbc_connection_pool_resources/_h2_pool_pool_numconncreated#count 0
......
......
```

Healthcheck Stats

"HealthCheck Services" can be exposed as **Custom Vendor Metrics** by the user by

supplying a custom metrics.xml file in the `domain-dir/config/` directory as well.

The following statistics are the integrated HealthCheck services metrics that are automatically exposed to the Microprofile Metrics service:

HealthCheck Service	Service ID	Attributes	Sub-Attributes
Stuck Threads	`healthcheck-stuck`	NA	`count`, `maxDuration`
Connection Pool	`healthcheck-cpool`	by default `H2Pool`, `__TimerPool`	`usedConnection`, `freeConnection`, `totalConnection`

Example

Here's an example of exposing the HealthCheck stats to the Microprofile Metrics endpoint:

```
<metadata>
    <name>thread.stuck.count</name>
    <service>healthcheck-stuck#count</service>
    <type>gauge</type>
    <unit>none</unit>
    <displayName>Stuck Thread Count</displayName>
    <description>Displays the stuck thread count which is blocked, and can't return
to the thread pool for a certain amount of time.</description>
</metadata>
```

Which will yield the following results:

```
# TYPE vendor_thread_stuck_count gauge
# HELP vendor_thread_stuck_count Displays the stuck thread count which is blocked,
and can't return to the threadpool for a certain amount of time.
vendor_thread_stuck_count 0
......
```

IMPORTANT | Health Check services need to be enabled first

Templating Metrics for Healthcheck Service

It is possible to create a template for a metric for multiple resource, attributes or sub-attributes configurations.

These placeholders are used as follows:

`attribute`: Used as a placeholder for the name of an **attribute** of an AMX MBean.

`subattribute`: Used as a placeholder for the name of a **sub-attribute** (or second-level attribute), which corresponds to the attribute of a `CompositeData` object.

IMPORTANT	You will need to use the `${attribute}` and `${subattribute}` placeholders in the `name` element as well to prevent name collisions between different templated metrics.

Example

Here's an example of defining a custom metric definition template with these placeholders that will generate a counter metric for each specific connection metric that is available to all JDBC connection pools:

```
<metadata>
    <name>connection.pool.${attribute}.${subattribute}</name>
    <service>healthcheck-cpool/${attribute}#${subattribute}</service>
    <type>counter</type>
    <unit>none</unit>
    <displayName>${attribute} ${subattribute}</displayName>
    <description>Displays the number of ${subattribute} in the Connection Pool
${attribute}.</description>
</metadata>
```

Which will yield the following results:

```
# TYPE vendor_connection_pool_H2Pool_freeConnection_total counter
# HELP vendor_connection_pool_H2Pool_freeConnection_total Displays the number of
freeConnection in the Connection Pool H2Pool.
vendor_connection_pool_H2Pool_freeConnection_total 0
# TYPE vendor_connection_pool_H2Pool_totalConnection_total counter
# HELP vendor_connection_pool_H2Pool_totalConnection_total Displays the number of
totalConnection in the Connection Pool H2Pool.
vendor_connection_pool_H2Pool_totalConnection_total 0
# TYPE vendor_connection_pool_H2Pool_usedConnection_total counter
# HELP vendor_connection_pool_H2Pool_usedConnection_total Displays the number of
usedConnection in the Connection Pool H2Pool.
vendor_connection_pool_H2Pool_usedConnection_total 0
# TYPE vendor_connection_pool___TimerPool_freeConnection_total counter
# HELP vendor_connection_pool___TimerPool_freeConnection_total Displays the number
of freeConnection in the Connection Pool __TimerPool.
vendor_connection_pool___TimerPool_freeConnection_total 0
# TYPE vendor_connection_pool___TimerPool_totalConnection_total counter
# HELP vendor_connection_pool___TimerPool_totalConnection_total Displays the number
of totalConnection in the Connection Pool __TimerPool.
vendor_connection_pool___TimerPool_totalConnection_total 0
```

```
# TYPE vendor_connection_pool___TimerPool_usedConnection_total counter
# HELP vendor_connection_pool___TimerPool_usedConnection_total Displays the number
of usedConnection in the Connection Pool __TimerPool.
vendor_connection_pool___TimerPool_usedConnection_total 0
......
```

Eclipse Microprofile Openapi API

Azul Payara 7.2.0 provides MicroProfile OpenAPI 3.1

Background

OpenAPI was designed to provide a standard interface for documenting and exposing RESTful APIs. The MicroProfile OpenAPI API provides a set of interfaces and annotations to allow application developers to produce an OpenAPI document from their JAX-RS applications. OpenAPI is somewhat similar to the Swagger API, so many concepts in the API will be familiar.

The complete specification can be found on the Eclipse MicroProfile website.

Breaking changes introduced in MicroProfile OpenAPI 3.1 are listed in the official specification under Incompatible Changes

IMPORTANT	Version 3.1 of the OpenAPI specification does not define the behaviour of implementations where multiple applications are deployed to the same container. This limitation will be addressed in a later version of the OpenAPI specification. Payara Server allows you to see multiple applications in the OpenAPI document, see multiple applications.

REST Endpoint

MicroProfile OpenAPI specifies that an OpenAPI document should be published at the endpoint `/openapi` documenting a deployed application. For both Payara Server and Payara Micro, this means that the endpoint can be found on any HTTP port (e.g. `8080`) after deploying an application.

WARNING	The OpenAPI document displayed will always be for the most

> recently deployed application, except in the special case of disabling the service.

By default, the OpenAPI document is displayed in YAML format, although this can be changed to JSON by changing the `Accept` header of the request. Alternatively if using a browser, this can also be done by specifying a `format` query parameter.

For example, `http://localhost:8080/openapi?format=json`. The valid values are `json` and `yaml`.

Openapi Configuration Sources

MicroProfile OpenAPI specifies several ways of configuring the data in the produced OpenAPI document. These are all provided by the OpenAPI API artifact:

```
<dependency>
    <groupId>org.eclipse.microprofile.openapi</groupId>
    <artifactId>microprofile-openapi-api</artifactId>
    <version>{mpOpenAPIVersion}</version>
    <scope>provided</scope>
</dependency>
```

Configuration Properties

The OpenAPI document generation process can be configured through the "Config API". The following are the standard configuration properties as presented by the specification:

Property key	Description
`mp.openapi.model.reader`	Configuration property to specify the fully qualified name of the OASModelReader implementation.
`mp.openapi.filter`	Configuration property to specify the fully qualified name of the OASFilter implementation.
`mp.openapi.scan.disable`	Configuration property to disable annotation scanning. Default value is `false`.

Property key	Description
`mp.openapi.scan.packages`	Configuration property to specify the list of packages to scan. For example, `mp.openapi.scan.packages=com.xyz.PackageA,com.xyz.PackageB`
`mp.openapi.scan.classes`	Configuration property to specify the list of classes to scan. For example, `mp.openapi.scan.classes=com.xyz.MyClassA,com.xyz.MyClassB`
`mp.openapi.scan.exclude.packages`	Configuration property to specify the list of packages to exclude from scans. For example, `mp.openapi.scan.exclude.packages=com.xyz.PackageC,com.xyz.PackageD`
`mp.openapi.scan.exclude.classes`	Configuration property to specify the list of classes to exclude from scans. For example, `mp.openapi.scan.exclude.classes=com.xyz.MyClassC,com.xyz.MyClassD`
`mp.openapi.servers`	Configuration property to specify the list of global servers that provide connectivity information. For example, `mp.openapi.servers=https://xyz.com/v1,https://abc.com/v1`
`mp.openapi.servers.path.`	Prefix of the configuration property to specify an alternative list of servers to service all operations in a path. For example, `mp.openapi.servers.path./airlines/ngs/{id}=https://xyz.io/v1`
`mp.openapi.servers.operation.`	Prefix of the configuration property to specify an alternative list of servers to service an operation. Operations that want to specify an alternative list of servers must define an `operationId`, a unique string used to identify the operation. For example, `mp.openapi.servers.operation.getBooking=https://abc.io/v1`

Property key	Description
`mp.openapi.schema.`	Prefix of the configuration property to specify an alternative list of servers to service an operation. Prefix of the configuration property to specify a schema for a specific class, in JSON format. The remainder of the property key must be the fully-qualified class name. The value must be a valid OpenAPI schema object, specified in the JSON format. The use of this property is functionally equivalent to the use of the `@Schema` annotation on a Java class, but may be used in cases where the application developer does not have access to the source code of a class. When a `name` key is provided with a string value, the schema will be added to the `schemas` collection in the `components` object of the resulting OpenAPI document using `name`'s value as the key. For example, in the case where an application wishes to represent a Java `Date` object in epoch-style milliseconds, the following configuration can be used (line escapes and indentation added for readability): `mp.openapi.schema.java.util.Date = { \ "name": "EpochMillis", \ "type": "number", \ "format": "int64", \ "description": "Milliseconds since January 1, 1970, 00:00:00 GMT" \ }`

In addition to the standard properties, the following properties have been added as extensions within Azul Payara:

Property key	Description
`mp.openapi.extensions.scan.lib`	Configuration property that allows the server to scan packaged JAR files inside the WAR's library directory (`WEB-INF/lib`). If the WAR is inside EAR, the server also scans JAR files inside the EAR's `lib/` directory. Default value is `false`.

Oasmodelreader

`org.eclipse.microprofile.openapi.OASModelReader`

The OASModelReader interface can be implemented by an application developer. It must then be declared as a Config API variable in the form:

```
mp.openapi.model.reader=fish.payara.examples.OASModelReaderImpl
```

This class will be called at the beginning of the OpenAPI document generation, and will create the initial OpenAPI document.

Oasfilter

`org.eclipse.microprofile.openapi.OASFilter`

The OASFilter interface can be implemented by an application developer. It must then be declared as a Config API variable in the form:

```
mp.openapi.filter=fish.payara.examples.OASFilterImpl
```

Each method in the implementation will be called on every appropriate element in the OpenAPI model. For example, the method `filterPathItem(PathItem pathItem)` will be called for every `PathItem` in the document. This class is called last, before the document is published.

Static Document

The MicroProfile OpenAPI supports using a static OpenAPI document to build from. This static file can be placed in either `META-INF` directory in a WAR file. Most often, this means putting the file in `src/main/resources/META-INF`.

The following file names are allowed for this file. The file given must also be in the specified format.

File Format	Allowed File Names
`yaml`	`openapi.yaml`, `openapi.yml`
`json`	`openapi.json`

Annotations

The MicroProfile OpenAPI API provides many annotations to use to augment the OpenAPI document. These are detailed in the OpenAPI Specification. These annotations are applied before the OASFilter.

Example

The following code could be used to give the corresponding operation an ID of "hello world".

```
import org.eclipse.microprofile.openapi.annotations.Operation;

@Path("/app")
public class AppResource{
    @GET
    @Operation(operationId = "hello world")
    public String helloWorld() {
        return "Hello World!";
    }
}
```

Here's an example of how the OpenAPI document of a sample application would look like:

```
openapi: 3.0.0
info:
  title: A Test Application
  version: "1.0"
servers:
- url: http://localhost:8080/openapi-example-1.0
  description: Default Server.
- url: https://localhost:8181/openapi-example-1.0
  description: Default Server.
paths:
  /api/hello:
    get:
      operationId: hello-world
      responses:
        default:
          content:
            text/plain:
              schema:
                type: string
          description: Default Response.
      deprecated: false
endpoints:
  /openapi-example-1.0:
  - /api/hello
components: {}
```

Deployed Endpoints

When more than one application is deployed, a merge is done in the OpenAPI document. The resulting document will contain all application and path definitions present. To improve the its readability, an `endpoints` attribute which lists all deployed endpoints grouped by their application context roots has been added to the document.

NOTE The `endpoints` attribute is an proprietary Azul Payara extension and it's not portable.

- **The following example shows an OpenAPI document generated when 2 different applications are deployed**:

```
openapi: 3.0.0
info:
  title: Deployed Resources
  version: 1.0.0
servers:
- url: http://localhost:8080/ClusteredSingleton
  description: Default Server.
- url: https://localhost:8181/ClusteredSingleton
  description: Default Server.
- url: http://localhost:8080/SimpleWAR
  description: Default Server.
- url: https://localhost:8181/SimpleWAR
  description: Default Server.
paths:
  /resources/javaee8:
    get:
      operationId: ping
      responses:
        default:
          content:
            '*/*':
              schema:
                type: object
          description: Default Response.
  /resources/randomNumberGen:
    get:
      operationId: randomNumberGen
      responses:
        default:
          content:
            '*/*':
              schema:
                type: integer
          description: Default Response.
  /rest/request:
    get:
      operationId: getXml
      responses:
```

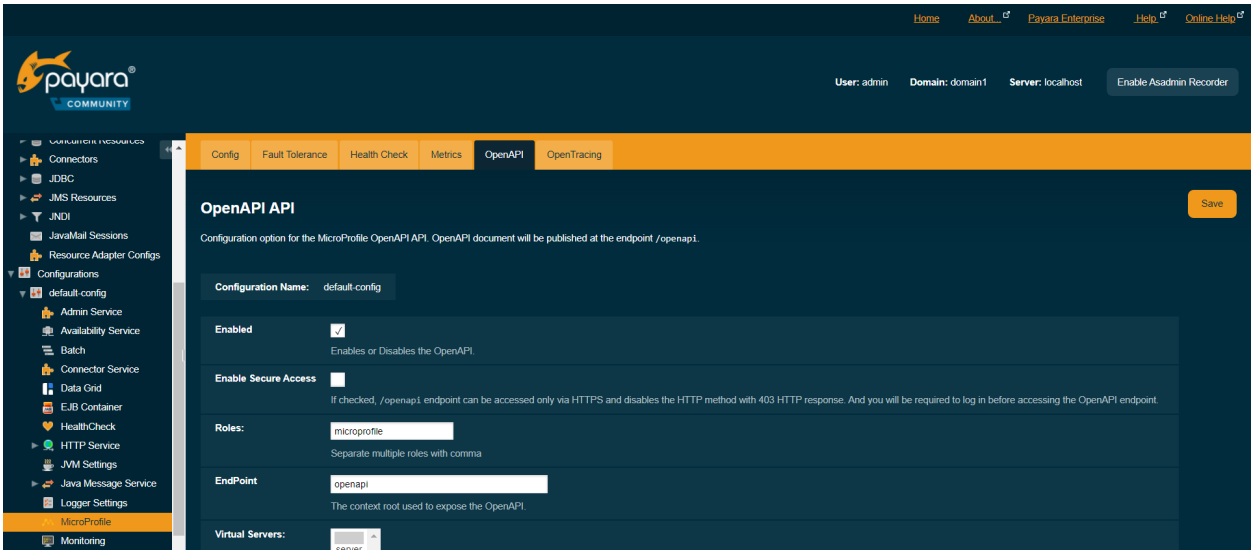
```
        default:
          content:
            text/plain:
              schema:
                type: string
          description: Default Response.
    put:
      operationId: putXml
      requestBody:
        content:
          application/xml:
            schema:
              type: string
      responses:
        default:
          content:
            '*/*':
              schema:
                type: object
          description: Default Response.
endpoints:
  /ClusteredSingleton:
  - /resources/javaee8
  - /resources/randomNumberGen
  /SimpleWAR:
  - /rest/request
components: {}
```

Openapi Configuration

On Payara Server, MicroProfile OpenAPI settings can be configured by using the Admin Console or Asadmin CLI commands.

Using the Admin Console

To configure the OpenAPI in the Admin Console, go to Configuration [instance-configuration (like `server-config`)] MicroProfile OpenAPI:



Using Asadmin Commands

Use the following commands to configure the settings of the MicroProfile OpenAPI service:

- set-openapi-configuration
- get-openapi-configuration

Security Configuration

By default, the OpenAPI endpoint binds to the root context application which is the `default-web-module` (also known as `docroot`) system application and `thedefault-web-module` application, which is secured under the default realm (`file`) of the server.

If a user application is deployed in the empty context-root, then the security configuration of this application will be shared by the OpenAPI endpoint, so exert extreme caution when making these changes.

Upgrading from Microprofile 3.x to 4.x

MicroProfile 4.0 brings with it a number of changes to MicroProfile OpenAPI, among these changes are some which break previous behaviour. The breaking changes are primarily to do with the removal of a number of methods previously marked as deprecated from the API, and only affect the use of OpenAPI programmatically (there are no breaking changes pertaining to the use of the annotations).

- `Scopes` interface removed - Use `Map<String, ServerVariable>`.
- `ServerVariables` interface removed - Use `Map<String, ServerVariable>`.
- The following interfaces no longer extend `Map` - Ensure you are not using any of the methods or functionality gained from the `Map` class:
  - `APIResponses`
    - `get(Object key)` - Use `getAPIResponse(String)`.
    - `containsKey(Object key)` - Use `hasAPIResponse(String)`.
    - `put(String key, PathItem value)` - Use `addAPIResponse(String, APIResponse)`.
    - `putAll(Map<? extends String, ? extends APIResponse> m)` - Use `setAPIResponses(Map)`.
    - `remove(Object key)` - Use `removeAPIResponse(String)`.
  - `Callback`
    - `get(Object key)` - Use `getPathItem(String)`.
    - `containsKey(Object key)` - Use `hasPathItem(String)`.
    - `put(String key, PathItem value)` - Use `addPathItem(String, PathItem)`.
    - `putAll(Map<? extends String, ? extends PathItem> m)` - Use `setPathItems(Map)`.
    - `remove(Object key)` - Use `removePathItem(String)`.
  - `Content`
    - `get(Object key)` - Use `getMediaType(String)`.
    - `containsKey(Object key)` - Use `hasMediaType(String)`.
    - `put(String key, MediaType value)` - Use `addMediaType(String, MediaType)`.

        - `putAll(Map<? extends String, ? extends MediaType> m)` - Use `setMediaTypes(Map)`.
        - `remove(Object key)` - Use `removeMediaType(String)`.
    - `Path`
        - `get(Object key)` - Use `getPathItem(String)`.
        - `containsKey(Object key)` - Use `hasPathItem(String)`.
        - `put(String key, PathItem value)` - Use `addPathItem(String, PathItem)`.
        - `putAll(Map<? extends String, ? extends PathItem> m)` - Use `setPathItems(Map)`.
        - `remove(Object key)` - Use `removePathItem(String)`.
    - `SecurityRequirement`
        - `get(Object key)` - Use `getScheme(String)`.
        - `containsKey(Object key)` - Use `hasScheme(String)`.
        - `put(String key, List<String> value)` - Use `addScheme(String, List<String>)`.
        - `putAll(Map<? extends String, ? extends List<String> m)` - Use `setSchemes(Map)`.
        - `remove(Object key)` - Use `removeScheme(String)`.
- A number of methods have been removed from model interfaces, in addition to those above:
    - OASFactory
        - `createScopes` - use `Map<String, String>`.
        - `createServerVariables` - use `Map<String, ServerVariable>`.
    - OAuthFlow

- `setScopes(Scopes scopes)` - Use `setScopes(Map)`.
- `scopes(Scopes scopes)` - Use `scopes(Map)`.

- OpenAPI
  - `path(String name, PathItem path)` - use `Paths#addPathItem(String, PathItem)` on `OpenAPI#getPaths`
- PathItem
  - `readOperations` - use `Map#values()` on `PathItem#getOperations()`.
  - `readOperationsMap` - use `getOperations()`.
- Schema
  - `getAdditionalProperties` - use `getAdditionalPropertiesSchema()` or `getAdditionalPropertiesBoolean()`.
  - `setAdditionalProperties(Schema additionalProperties)` - use `setAdditionalPropertiesSchema(Schema)`.
  - `setAdditionalProperties(Boolean additionalProperties)` - use `setAdditionalPropertiesBoolean(Boolean)`.
  - `additionalProperties(Schema additionalProperties)` - use `additionalPropertiesSchema(Schema)`.
  - `additionalProperties(Boolean additionalProperties)` - use `additionalPropertiesBoolean(Boolean)`.
- Server
  - `setVariables(ServerVariables variables)` - Use `setVariables(Map)`.
  - `variables(ServerVariables variables)` - Use `variables(Map)`.

Opentelemetry and Opentracing Support

Azul Payara natively supports OpenTelemetry APIs as well as export of traces over the OTLP protocol. Natively in this context means, that deployment of an OpenTelemetry

Agent is not necessary and may lead to duplicate traces.

All OpenTracing support in Azul Payara is done by means of the OpenTracing Compatibility Layer.

Azul Payara satisfies requirements of both MicroProfile OpenTracing 3.0 and MicroProfile Telemetry Tracing 1.1 provided appropriate Span Convention Settings are applied.

IMPORTANT	OpenTelemetry implementation only covers the tracing aspect of OpenTelemetry, no metrics or logs are provided by default. Manual instrumentation is needed to enable these.

Enabling Opentelemetry

OpenTelemetry is enabled for a particular application when:

- It defines the MicroProfile configuration property `otel.sdk.disabled` with a value of `false` as defined by MicroProfile Telemetry Tracing
- The Request Tracing Service is enabled
- The `otel.sdk.disabled` System property is set to `false`
- The `OTEL_SDK_DISABLED` Environment variable is set to `false`

Configuring Opentelemetry

When OpenTelemetry is enabled it proceeds to configure OpenTelemetry components by means of autoconfiguration, using the application's MicroProfile Config properties prefixed with `otel.` first, but also falling back to system properties and environment variables as described in the linked documentation.

If the configured component does not ship with Payara distribution, it can be added as a server or application library as described in Using Additional Components.

Default Settings

With no other configuration properties specified OpenTelemetry components will:

- Accept and propagate trace headers and baggage according to W3C recommendations Trace Context and Baggage
- Export traces via OTLP protocol to `http://localhost:4317`
- Respect parent trace's sampling decision and always sample local root traces (sampler setting `parentbased_always_on`)
- Will not export metrics or logs.

To easily receive traces in default configuration run the following Jaeger container and then access its UI at port `16686`:

```
docker run --name jaeger \
   -e COLLECTOR_OTLP_ENABLED=true \
   -p 16686:16686 \
   -p 4317:4317 \
   -p 4318:4318 \
   jaegertracing/all-in-one:latest
```

Span Convention Settings

In order to improve consistency among products our implementation offers several strategies of naming and tagging spans of Jakarta REST client and server calls.

These are configured by specifying the MicroProfile Config property `payara.telemetry.span-convention` for your application.

The following values are accepted:

- `opentelemetry` follows convention prescribed by OpenTelemetry semantic conventions for HTTP spans (default setting)
- `opentracing-http-path` satisfies MicroProfile OpenTracing setting for Operation Name Provider `http-path`
- `opentracing-class-method` corresponds to MicroProfile OpenTracing setting for Operation Name Provider `class-method`
- `microprofile-telemetry` satisfies span naming as tested by the MicroProfile Telemetry Tracing TCK

Span Name Examples

Let's assume `GET` request made to `/app/rest/resource/method`, where

- `/app` is application context root
- `/rest` is `@ApplicationPath` of Jakarta REST
- `/resource` is `@Path` annotation of class `example.Resource`
- `/method` is `@Path` annotation of method `restMethod`

The following span names will be created for this request for different Span Convention settings:

Span-Convention Setting	Server Span Name
`opentelemetry`	`GET /app/rest/resource/method`
`opentracing-class-method`	`GET:example.Resource.restMethod`
`opentracing-http-path`	`GET:/resource/method`
`microprofile-telemetry`	`/app/rest/resource/method`

Suppressing Export Warning Message

When the export endpoint is not available in default setting the following error message is logged by logger `io.opentelemetry.exported.internal.grpc.OkHttpGrpcExporter`:

```
Failed to export spans. The request could not be executed. Full error message:
Failed to connect to `localhost/127.0.0.1:4317`
```

If you don't intend to run an **OTEL** endpoint on `localhost` or at all you need to configure the exporter by means of MP Config Properties, system properties, or environment variables:

- To turn off exports, set the `otel.traces.exporter` property to `none`
- To set the remote endpoint <code>otel.exporter.otlp.endpoint</code> to <code><a href="http://<collector-host>:<port&gt" class="bare">http://<collector-host>:<port&gt;</code>

Manual Tracing

To fine-tune spans and have better control about their boundaries you can use directly either the OpenTracing or OpenTelemetry APIs.

Dependencies

The appropriate versions of APIs are provided by "Payara BOM artifact" matching the version of the Azul Payara distribution you are using.

Azul Payara provides the following APIs, and it is advised to mark them as `provided` dependencies in your application and don't package them with it:

- `io.opentelemetry:opentelemetry-sdk-extension-autoconfigure`
- `io.opentelemetry:opentelemetry-sdk`
- `io.opentelemetry:opentelemetry-sdk-common`
- `io.opentelemetry:opentelemetry-sdk-metrics`
- `io.opentelemetry:opentelemetry-sdk-logs`
- `io.opentelemetry:opentelemetry-api-logs`
- `io.opentelemetry:opentelemetry-sdk-extension-autoconfigure-spi`
- `io.opentelemetry:opentelemetry-semconv`
- `io.opentelemetry:opentelemetry-opentracing-shim`
- `io.opentelemetry:opentelemetry-api`
- `io.opentelemetry:opentelemetry-context`
- `io.opentracing:opentracing-api`
- `io.opentracing:opentracing-noop`
- `io.opentelemetry:opentelemetry-exporter-otlp`
- `io.opentelemetry:opentelemetry-sdk-trace`
- `io.opentelemetry:opentelemetry-exporter-otlp-common`
- `io.opentelemetry:opentelemetry-exporter-common`

Using Opentelemetry Manually

```
import io.opentelemetry.api.trace.Tracer;

@RequestScoped
public class TracedComponent {

    @Inject
    Tracer tracer;

    public void tracedMethod() {
        var span = tracer.spanBuilder("tracedTask").startSpan();

        try (var scope = span.makeCurrent()) {
            // do work, add information to span
            if (span.isRecording()) {
                // compute expensive events or tags
                // only if the span is being sampled
            }
        } finally {
            span.setStatus(StatusCode.OK);
            span.end();
        }
    }
}
```

Using Opentracing Manually

```
import io.opentracing.Tracer;

@RequestScoped
public class TracedComponent {

    @Inject
    Tracer tracer;

    public void tracedMethod() {
        var span = tracer.buildSpan("tracedTask").start();

        try (var scope = tracer.activateSpan(span)) {
            // do work, add information to span
        } finally {
            span.finish();
        }
    }
}
```

Using Additional Components

If your application requires OpenTracing components that are not shipped with Azul Payara, it is possible to either put them in common libraries, or ship them with an application. Extension components can be even coded directly in the application code.

Provider Components in Library Directory

NOTE	Applies to Payara Server only.

Autoconfiguration can pick up components, which are placed in directory `domain_dir/lib` for example by means of the `asadmin add-library` command.

Example: Adding Export to Log

1. Download opentelemetry-exporter-logging.jar
2. Run `asadmin add-library opentelemetry-exporter-logging.jar`
3. Use the component by defining `otel.traces.exporter=logging`

Provider Components in Application Code

Applications can declare and use their own OpenTelemetry components such as exporters or samplers by writing against autoconfiguration SPI and placing appropriate Service Loader resource in the application

Example: Minimal Logging Exporter

src/main/resources/META-INF/services/io.opentelemetry.sdk.autoconfigure.spi.traces.ConfigurableSpanExporterProvider

```
fish.payara.example.LogExporter.Provider
```

src/main/java/fish/payara/example/LogExporter.java

```
public class LogExporter implements SpanExporter {

    private static final Logger LOGGER = Logger.getLogger(LogExporter.class.
getName());

    @Override
    public CompletableResultCode export(Collection<SpanData> spans) {
        spans.forEach(s -> LOGGER.info(s.toString()));
        return CompletableResultCode.ofSuccess();
    }

    @Override
    public CompletableResultCode flush() {
        return CompletableResultCode.ofSuccess();
    }

    @Override
```

```
    public CompletableResultCode shutdown() {
        return CompletableResultCode.ofSuccess();
    }

    // This is registered as SPI and creates configured exporter
    public static class Provider implements ConfigurableSpanExporterProvider {

        @Override
        public SpanExporter createExporter(ConfigProperties configProperties) {
            return new LogExporter();
        }

        @Override
        public String getName() {
            return "logs";
        }
    }
}
```

Relation to Request Tracing Service

All spans created by OpenTelemetry and OpenTracing APIs are passed into Request Tracing Service when they end. Propagated trace IDs are propagated into Request Tracing Service at the start of spans.

Sampling decisions of Request Tracing service are ***not*** taken into account when these traces are created, because the Request Tracing Service makes a tracing decision at the end of span rather than at the beginning.

This means that traces might get exported via **OTLP** but not passed to the notifiers configured in the Request Tracing Service, because the service might have taken the decision not to sample this particular trace.

Information that server's components put into Request Tracing Service is not available to OpenTelemetry traces, as information does not flow in this direction.

However, more and more parts of Payara Server will gradually switch to using OpenTelemetry API natively and therefore this gap will eventually close.

Eclipse Microprofile Opentracing

Azul Payara 7.2.0 provides MicroProfile OpenTracing 3.0

Background

The Open Tracing API was created to allow easier tracing of the flow of a request in microservices environment across service boundaries.

The complete specification can be found on the Eclipse MicroProfile website.

Breaking changes introduced in MicroProfile OpenTracing 3.0 are listed in the official specification under Incompatible Changes

Support for MicroProfile Open Tracing is provided via compatibility layer for OpenTelemetry.

To achieve full compatibility in naming the spans with the spec a span convention needs to be configured as described in Span Naming Convention.

Text and Binary Carriers Support

The Azul Payara implementation of the `io.opentracing.Tracer` interface supports the following carriers:

- `TextMap` carriers with `HTTP_HEADERS` and `TEXT_MAP` formats

JAX-WS Support

Azul Payara's MicroProfile implementation supports tracing of requests that are handled by SOAP Web Services implemented via the JAX-WS API. Since JAX-WS is only supported on Payara Server Full Profile, this support is not covered on Payara Micro.

Opentracing Configuration

Span Naming Convention

By default, span names for JAX-RS endpoints follow recommendations set forth by OpenTelemetry Semantic Conventions. To have span names follow naming conventions defined by MicroProfile OpenTelemetry, following MicroProfile Config property must be set for application or globally for the server or domain:

```
payara.telemetry.span-convention=opentracing-class-method
```

All valid values for the property are documented in span-convention-settings

Using Alternate Opentracing Implementations

Currently, it is not possible to use other OpenTracing implementations in Azul Payara.

It is however possible to deploy and configure other OpenTelemetry exporters (i.e. Zipkin) into the server or ship them with application.

Please consult using-additional-components.

Eclipse Microprofile Rest Client API

Azul Payara 7.2.0 provides MicroProfile Rest Client 3.0.1

Background

The MicroProfile Rest Client provides a type-safe approach to invoke RESTful services over HTTP. The aim of this specification is to provide a much more natural coding style, with the underlying MicroProfile implementation handling the communication between the client and service.

The complete specification can be found on the Eclipse MicroProfile website.

Breaking changes introduced in MicroProfile Rest Client 3.0.1 are listed in the official specification under Release Notes for MicroProfile Rest Client 3.0.1

NOTE	The Azul Payara implementation of this specification is based on Jersey's MicroProfile Rest client, the interaction with client interfaces is translated into interaction with standard JAX-RS clients.

Jakarta CDI

Rest Client interfaces may be injected as CDI beans. To do this, you must first register your interface class by using the `@RegisterRestClient` annotation, and then use the `@RestClient` qualifier on the injected bean, like so:

```
@RegisterRestClient
public interface MyServiceClient {
    @GET
```

```
    @Path("/greet")
    Response greet();
}

@ApplicationScoped
public class MyService {
    @Inject
    @RestClient
    private MyServiceClient client;
}
```

The endpoint to connect to in such a case must either be configured directly in attribute `baseUri` of the `@RegisterRestClient` annotation or via Microprofile Configuration properties.

Asynchronous Communication

It is possible for Rest Client interface methods to be declared asynchronous, by having return type of `CompletionStage<?>`. This allows the thread invoking the interface method to proceed while the effective client communication occurs on another thread.

The thread pool used in this call can be configured via the `RestClientBuilder.executorService` method. If this setting is not configured, a default, server-managed thread pool will be provided.

Providers

The `RestClientBuilder` interface extends the `Configurable` interface of the JAX-RS API, allowing users to register custom providers while at the object's instantiation. The following provider types are supported:

- `ClientRequestFilter`
- `ClientResponseFilter`
- `MessageBodyReader`
- `MessageBodyWriter`
- `ParamConverter`
- `ReaderInterceptor`
- `WriterInterceptor`

- `ResponseExceptionMapper`

See the full documentation (linked below) for more information on available registration methods.

SSL Configuration

On a per-client basis, or via Microprofile Configuration properties, the following aspects of secure client communication can be set:

- Truststore, listing trusted certificates and authorities — by default the one used by Azul Payara is used
- Keystore for setting up mutual SSL trust
- Hostname verifier, useful for suppressing validity checks for whitelisted host names

Programmatically it is also possible to provide an implementation of the `javax.net.ssl.SSLContext` class to have major control over all aspects of this secure communication channel.

SSL Context Configuration

You can define the certificate alias rest client will use with one of the following approaches:

During REST Client Call

Property	Public API Constant	Description
`fish.payara.jaxrs.client.certificate.alias`	`PayaraConstants.REST_CLIENT_CERTIFICATE_ALIAS`	The alias name of the certificate

NOTE The name of the property is defined as a constant in the `fish.payara.security.client.PayaraConstants` class.

Example

Here's an example of how to set up the property when executing a new MP Rest Client call:

```
public class Component{
    public void executeRestClientCall(){
        RestClientBuilder.newBuilder()
                         .baseUri(new URI("https://localhost:8080/service"))
                         .property(PayaraConstants.REST_CLIENT_CERTIFICATE_ALIAS,
"someAliasName")
                         .build(Service.class);
    }
}
```

On Microprofile Configuration

Property	Public API Constant	Description
`payara.certificate.alias`	`PayaraConstants.MP_CONFIG_CLIENT_CERTIFICATE_ALIAS`	The alias name of the certificate

You can define the alias globally as a "MicroProfile Config" property, either in an application-specific config source only for a single application, or with a global config source for all applications. In this case you don't need to add it during every single Rest Client call.

NOTE For this to work, you need to include the certificate with the given alias in the default Payara Server keystore.

Proxy Configuration

MicroProfile Rest Client 2.0 introduced support for configuring proxy servers via MicroProfile Config properties and the `RestClientBuilder` API. On a vanilla installation however, Azul Payara will not use these settings thus they will be ignored. This is due to Jersey, the JAX-RS client and underlying implementation of MicroProfile Rest Client, which by default, doesn't support the configuration of proxies on a per-client basis (only via a global JVM-level system property).

However, Jersey supports proxy configuration on a per-client basis when using non-default "connectors" (the means by which Jersey performs the actual network call)

If you wish to make use of this feature, you must perform a number of steps to configure Payara Server to use of one of these non-default Jersey connectors.

Below are instructions for how to configure Payara Server to make use of Apache HTTP Client connector.

Create a Restclientlistener

To configure Jersey to use Apache HTTP Client as its connector, a `RestClientListener` must be used to register the connector for each new client. You can add one to your application by simply registering as a custom JAX-RS provider.

Below is a simple example of registering the Jersey Apache HTTP Client Connector:

```
public class RestClientApacheHttpClientListener implements RestClientListener {

    @Override
    public void onNewClient(Class<?> aClass, RestClientBuilder restClientBuilder) {
        restClientBuilder.register(new ApacheConnectorProvider());
    }
}
```

NOTE

The `ApacheConnectorProvider` class can be found in the `org.glassfish.jersey.connectors:jersey-apache-connector` library, please refer to the Payara BOM artefact for the specific version of the Jersey connector to use.

Add Apache HTTP Client Dependencies

In addition to the above, you will also need to add the following dependencies to the Payara Server domain (if applicable):

- `org.apache.httpcomponents:httpclient-osgi:4.5.13`
- `org.apache.httpcomponents:httpcore-osgi:4.4.14`
- `commons-logging:commons-logging:1.2`

These can be included with your application or added to the server via the `add-library` command:

```
asadmin add-library httpclient-osgi-4.5.13.jar httpcore-osgi-4.4.14.jar commons-logging-1.2.jar
```

If you haven't bundled the library in your application, you will also need to add the `org.glassfish.jersey.connectors:jersey-apache-connector` dependency to Payara Server:

```
asadmin add-library jersey-apache-connector-${jersey.version}.jar
```

See Also

- {repo}/spec/src/main/asciidoc/clientexamples.asciidoc[MicroProfile Rest Client Examples]
- {repo}/spec/src/main/asciidoc/cdi.asciidoc[MicroProfile Rest Client CDI Integration]
- {repo}/spec/src/main/asciidoc/async.asciidoc[MicroProfile Rest Client Asynchronous Configuration]
- {repo}/spec/src/main/asciidoc/providers.asciidoc[MicroProfile Rest Client JAX-RS Providers]
- {repo}/spec/src/main/asciidoc/ssl.asciidoc[MicroProfile Rest Client SSL Configuration Specifics]

Eclipse Microprofile Telemetry

Azul Payara 7.2.0 provides MicroProfile Telemetry 1.1.

The aim of the spec is to adopt OpenTelemetry into MicroProfile applications, so they can benefit from distributed tracing.

OpenTelemetry thus becomes the primary observability API in Azul Payara.

OpenTelemetry defines more types of signals. As of version 7.2.0 and in line with MicroProfile spec, support is only provided for Tracing aspects for the API.

For more details about configuration of OpenTelemetry refer to "OpenTelemetry documentation" for application development.

Ecosystem

Azul Payara Ecosystem

Azul Payara not only provides application servers in the form of Payara Server and Payara Micro; there are also satellite projects which complement and enhance the developer experience.

These include:

"Payara Micro Maven Plugin"
A plugin for Apache Maven is available for Payara Micro.

"Payara Micro Maven Archetype"
An archetype for Apache Maven is available for Payara Micro.

"Payara Micro Gradle Plugin"
A plugin for Gradle is available for Payara Micro.

"Payara Starter"
An archetype is available for Azul Payara.

"Payara Tools Eclipse Plugin"
A plugin for using Payara is available for Eclipse.

"Payara Tools for Apache NetBeans"
The tools for using Payara are available in the Apache NetBeans.

"Payara IntelliJ IDEA Plugin"
A plugin for using Payara is available for IntelliJ IDEA.

"Payara Visual Studio Code Extension"
An extension for using Payara is available for Visual Studio Code.

"Cloud Connectors"
Cloud Connectors are a collection of JCA adapters which can interact with popular message brokers, including:

- "Amazon SQS"
- "Apache Kafka"
- "Azure Service Bus"
- "MQTT"

"Arquillian Containers"
Arquillian containers help perform integration tests for application servers. There are various containers available for Payara:

- "Payara Micro Managed"
- "Payara Server Remote"
- "Payara Server Managed"
- "Payara Server Embedded"

Maven Bill of Materials Artifact

Azul Payara consists of many artifacts and it depends on lots of APIs as well as their implementations. Having matching versions of these dependencies will prevent classloading issues at runtime. To this effect, Azul Payara's Bill Of Materials (BOM) artifact collects all of these versions for the following categories:

- Azul Payara Distribution artifacts
- Azul Payara Client libraries
- Jakarta EE, Eclipse Microprofile and OSGi APIs
- Arquillian Container Adapters
- Implementations and libraries included in Payara Server (see the list below)

Quick Start

To make use of the BOM, you need to list the artifact in `dependencyManagement` section of your project in scope `import`. Some of the artifacts are Payara-specific patch versions which are available in our repository payara-artifacts.

Add following snippet to your Maven project:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>fish.payara.api</groupId>
            <artifactId>payara-bom</artifactId>
            <version>7.2.0</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<!-- Some of the referenced APIs in POM are Payara's patched versions.
      These are published in following repo: -->
<repositories>
    <repository>
        <id>payara-patched-externals</id>
        <name>Payara Patched Externals</name>
        <url>https://nexus.payara.fish/repository/payara-artifacts/</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>

</repositories>
```

An example application utilizing the BOM can be seen in repository Payara Examples/. See the ecosystem/payara-bom/bom-import/ subproject for more details.

BOM Contents

The following is a summary of all dependencies included in the BOM, grouped by category.

Azul Payara Distributions

Distribution artifacts do not define a Maven scope and are therefore imported in `compile` scope.

Artifact	Example usage	Note
Payara Server	`<dependency> <groupId> fish.payara.distributions</groupId> <artifactId>payara</artifactId> <type>zip</type> </dependency>`	Also available are `payara-ml`, `payara-web`, `payara-web-ml`
Payara Micro	`<dependency> <groupId> fish.payara.extras</groupId> <artifactId>payara-micro</artifactId> </dependency>`	
Payara Embedded	`<dependency> <groupId> fish.payara.extras</groupId> <artifactId>payara-embedded-all</artifactId> </dependency>`	Also available `payara-embedded-web`
EJB HTTP Client	`<dependency> <groupId> fish.payara.extras</groupId> <artifactId>ejb-http-client</artifactId> </dependency>`	
Appclient	`<dependency> <groupId>fish.payara.server.appclient</groupId> <artifactId>payara-client</artifactId> </dependency>`	
Payara Public API	`<dependency> <groupId>fish.payara.api</groupId> <artifactId>payara-api</artifactId> </dependency>`	

Standard APIs

Standard APIs are defined in `provided` scope, as they already include classes that are present in the server's runtime.

Artifact	Example usage	Note

Jakarta EE	`<dependency>`
	`  <groupId>jakarta.platform</groupId>`
	`  <artifactId>jakarta.jakartaee-api</artifactId>`
	`</dependency>`
		Also available `jakarta.jakartaee-web-api`, and all individual API artifacts.

Eclipse Microprofile	`<dependency>`
	`  <groupId>org.eclipse.microprofile</groupId>`
	`  <artifactId>microprofile</artifactId>`
	`  <type>pom</type>`
	`</dependency>`
		Individual APIs are also declared.

OSGi	`<dependency>`
	`  <groupId>org.osgi</groupId>`
	`  <artifactId>osgi.core</artifactId>`
	`</dependency>`
		Also available are `org.cmpn`, `osgi.annotation`, `org.osgi.dto`, and `osgi.enterprise`

Test Frameworks and Other Tools

Arquillian is the recommended way to run integration tests written for Payara Server. Azul Payara provides multiple connectors fitting the deployment scenarios. These artifacts are declared in scope `test`.

Payara Server includes the H2 database, so their matching versions are also declared, but in their default scope.

Artifact	Example usage

Arquillian container - Remote	`<dependency>`
	`  <groupId>fish.payara.arquillian</groupId>`
	`  <artifactId>arquillian-payara-server-remote</artifactId>`
	`</dependency>`

Arquillian container - Embedded	`<dependency>`
	`  <groupId>fish.payara.arquillian</groupId>`
	`  <artifactId>arquillian-payara-server-embedded</artifactId>`
	`</dependency>`

Arquillian container - Managed	`<dependency>`
	`  <groupId>fish.payara.arquillian</groupId>`
	`  <artifactId>arquillian-payara-server-managed</artifactId>`
	`</dependency>`

Artifact	Example usage

Arquillian container - Payara Micro Managed	`<dependency>`
	`  <groupId>fish.payara.arquillian</groupId>`
	`  <artifactId>arquillian-payara-micro-managed</artifactId>`
	`</dependency>`

Arquillian container - Payara Micro Remote	`<dependency>`
	`  <groupId>fish.payara.arquillian</groupId>`
	`  <artifactId>arquillian-payara-micro-remote</artifactId>`
	`</dependency>`

Arquillian container - Payara Micro Remote Deployer	`<dependency>`
	`  <groupId>fish.payara.arquillian</groupId>`
	`  <artifactId>payara-micro-deployer</artifactId>`
	`</dependency>`

Payara Arquillian Client Dependencies	`<dependency>`
	`  <groupId>fish.payara.arquillian</groupId>`
	`  <artifactId>payara-client-ee9</artifactId>`
	`</dependency>`

H2 Database	`<dependency>`
	`  <groupId>com.h2database</groupId>`
	`  <artifactId>h2</artifactId>`
	`  <scope>test</scope>`
	`</dependency>`

Libraries and API Implementations

This is a non-exhaustive list of dependencies. By default all are declared in their respective default scopes, but it is more correct to use them with scope `provided` if they are already present in the server's internal modules.

Artifact	Example usage	Note

Jersey	`<dependency>`
	`  <groupId>org.glassfish.jersey.core</groupId>`
	`  <artifactId>jersey-client</artifactId>`
	`  <scope>test</scope>`
	`</dependency>`
	`<dependency>`
	`  <groupId>org.glassfish.jersey.inject</groupId>`
	`  <artifactId>jersey-hk2</artifactId>`
	`  <scope>test</scope>`
	`</dependency>`
		Example: JAX-RS clients in tests. All artifacts of the Jersey BOM are imported.

Artifact	Example usage	Note
Hibernate Validator	`<dependency> <groupId> org.hibernate.validator</groupId> <artifactId>hibernate-validator</artifactId> <scope>test</scope> </dependency>`	
EclipseLink - JPA Metamodel generator	`<dependency> <groupId> org.eclipse.persistence</groupId> <artifactId>org.eclipse.persistence.jpa.modelgen.processor</artifactId> </dependency>`	The only dependency that is `provided` scoped, as it is an annotation processor.
EclipseLink	`<dependency> <groupId> org.eclipse.persistence</groupId> <artifactId>org.eclipse.persistence.jpa</artifactId> <scope>provided</scope> </dependency>`	
Hazelcast	`<dependency> <groupId>com.hazelcast</groupId> <artifactId>hazelcast</artifactId> <scope>provided</scope> </dependency>`	The `hazelcast-kubernetes` dependency is also available.
Jackson	`<dependency> <groupId>com.fasterxml.jackson.datatype</groupId> <artifactId>jackson-datatype-jdk8</artifactId> </dependency>`	All artifacts of the Jackson BOM are imported.
Yasson	`<dependency> <groupId>org.eclipse</groupId> <artifactId>yasson</artifactId> <scope>test</scope> </dependency>`	

Artifact	Example usage	Note
Tyrus	`<dependency> <groupId> org.glassfish.tyrus</groupId> <artifactId>tyrus-client</artifactId> <scope>test</scope> </dependency> <dependency> <groupId> org.glassfish.tyrus</groupId> <artifactId>tyrus-container-grizzly-client</artifactId> <scope>test</scope> </dependency>`	

Maven Plugin

Payara Maven Plugins

The Payara Maven Plugins provide seamless integration with Maven, enabling developers to start, stop, and deploy Jakarta EE applications on Payara Server and Payara Micro. These plugins streamline development, support automated deployment, and enhance workflow efficiency.

Plugins

- "Payara Server Maven Plugin" - The Payara Server Maven Plugin allows developers to start, stop, and deploy applications to Payara Server directly from Maven, streamlining the development and deployment process.
- "Payara Micro Maven Plugin" - The Payara Micro Maven Plugin enables seamless integration with Payara Micro, providing goals to start, stop, and package lightweight, microservice-based Jakarta EE applications.

Payara Server Maven Plugin

Payara Server provides a Maven plugin that can start and stop the server and deploy Jakarta EE applications.

The plugin requires JDK 17 or higher.

Configurable Goals

`start`

Aim: This goal starts Payara Server with the specified configuration. **Usage**: `mvn`

`payara-server:start`

Example:

```
<plugin>
    <groupId>fish.payara.maven.plugins</groupId>
    <artifactId>payara-server-maven-plugin</artifactId>
    <version>${payara.server.maven.plugin.version}</version>
    <executions>
        <execution>
            <goals>
                <goal>start</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <payaraServerVersion>7.2025.1</payaraServerVersion>
        <javaHome>/path/to/Java/Executable</javaHome>
        <contextRoot>myApp</contextRoot>
    </configuration>
</plugin>
```

Common Configuration:

The following properties allow customization of local and remote deployments:

Property Name	System Property	Environment Variable	Default Value	Description
contextRoot	payara.context.root	PAYARA_CONTEXT_PATH	*none*	Defines the context root of an application.
adminPort	payara.admin.port	PAYARA_ADMIN_PORT	*none*	Admin port for Payara server administration.
httpPort	payara.http.port	PAYARA_HTTP_PORT	*none*	HTTP port for accessing deployed applications.
httpsPort	payara.https.port	PAYARA_HTTPS_PORT	*none*	HTTPS port for accessing deployed applications securely.
protocol	payara.protocol	PAYARA_PROTOCOL	*none*	Protocol to be used for server communication (e.g., HTTP or HTTPS).

Property Name	System Property	Environment Variable	Default Value	Description
adminPassword	payara.admin.password	PAYARA_ADMIN_PASSWORD	*none*	Password for the Payara admin user.
adminPasswordFile	payara.admin.password.file	PAYARA_ADMIN_PASSWORD_FILE	*none*	Path to the file containing the Payara admin password.
adminUser	payara.admin.user	PAYARA_ADMIN_USER	*none*	Username for Payara server administration.
instanceName	payara.instance.name	PAYARA_INSTANCE	*none*	Name of the Payara server instance.
daemon	payara.daemon	PAYARA_DAEMON	false	Starts Payara Server in separate JVM process and continues with the maven build.
immediateExit	payara.immediate.exit	PAYARA_IMMEDIATE_EXIT	false	If Payara Server is executed in daemon mode, the executor thread will wait for the ready message before shutting down its process. By setting `immediateExit` to `true` you can skip this and instantly interrupt the executor thread.
autoDeploy	payara.auto.deploy	PAYARA_AUTO_DEPLOY	false	Enables automatic compilation and deployment upon saving files.
keepState	payara.keep.state	PAYARA_KEEP_STATE	false	Persists session state across multiple redeployments.
liveReload	payara.live.reload	PAYARA_LIVE_RELOAD	false	Triggers a browser refresh for an up-to-date view after each redeployment.

Property Name	System Property	Environment Variable	Default Value	Description
browser	payara.browser	PAYARA_BROWSER	System default browser	Specifies the browser used for live reload. If not explicitly set, Payara selects the best available browser based on the system: Chrome (if installed), then Firefox, followed by the system default—Edge on Windows, Safari on macOS, and Firefox on Linux.
trimLog	payara.trim.log	PAYARA_TRIM_LOG	false	Refactors log format for improved readability.
hotDeploy	payara.hot.deploy	PAYARA_HOT_DEPLOY	false	Enables the Hot Deploy mode.
webappDirectory			${project.build.directory}/${project.build.finalName}	The directory where the web application is built.
debug	payara.debug	PAYARA_DEBUG	false	Enables debugging mode. If true, the server waits for a debugger to attach.
debugPort	payara.debug.port	PAYARA_DEBUG_PORT	*none*	The port for remote debugging.
httpConnectionTimeout	payara.http.connection.timeout	PAYARA_HTTP_CONNECTION_TIMEOUT	3000	Socket connection timeout (in milliseconds).
httpReadTimeout	payara.http.read.timeout	PAYARA_HTTP_READ_TIMEOUT	3000	Socket read timeout (in milliseconds).

Local Configuration:

The following properties allow customization of deployments to local instance.

Property Name	System Property	Environment Variable	Default Value	Description
javaHome	payara.java.home	PAYARA_JAVA_HOME	Environment Variable `JAVA_HOME` or `JDK_HOME`	Absolute path to the java executable.
payaraServerVersion	payara.server.version	PAYARA_SERVER_VERSION	`6.2025.6`	By default, start mojo fetches payara-server with version `6.2025.6`.
payaraServerPath	payara.server.path	PAYARA_SERVER_PATH	*none*	Absolute path to the Payara server installation directory.
artifactItem			*none*	Defines payara-server artifact with its coordinates. Specified artifact should be available in local maven repository.
domainName	payara.domain.name	PAYARA_DOMAIN_NAME	`domain1`	Payara server domain to be used.
exploded	payara.exploded	PAYARA_EXPLODED	false	If true, deploys the application in an exploded (unpacked) format.
commandLineOptions			*none*	Defines a list of command line options that will be passed onto payara-server.

Property Name	System Property	Environment Variable	Default Value	Description
javaCommandLine Options			*none*	Defines a list of command line options that will be passed to `java` executable. Command line options can either be defined as key-value pairs or just as list of values. key-value pairs will be formatted as `key=value`.

Remote Configuration:

The following properties allow customization of deployments to remote instance.

Property Name	System Property	Environment Variable	Default Value	Description
remote	payara.remote	PAYARA_REMOTE	false	If true, connects to a remote Payara server instance.
hostName	payara.host.name	PAYARA_HOST_NAME	*none*	Hostname or IP address of the Payara server.

`dev`

Aim: The dev goal is designed to streamline development by enabling `exploded`, `autoDeploy`, `liveReload`, `keepState`, and `trimLog` to true, which facilitates automatic compilation and deployment when saving files, resulting in faster iterations and a more efficient development workflow.

Usage:

```
mvn payara-server:dev
```

The `dev` goal is an extended goal of `start` and is pre-configured for developing web applications in development mode which is equivalent to starting the Payara Server instance using the following command:

```
mvn payara-server:start -DautoDeploy=true -DliveReload=true -Dexploded=true
-DkeepState -DtrimLog=true
```

Features in Dev Mode

1. **Auto Deploy**: The AutoDeploy feature enables automatic compilation and deployment of the application upon saving files within the project structure. Enabled by setting `autoDeploy=true`.

   The AutoDeploy feature relies on Java's WatchService to monitor changes in source directories, enabling continuous build and reload using Maven's Invoker. It dynamically modifies Maven goals based on file modifications, supports asynchronous execution for concurrent build tasks, and includes specifics for tracking changes in source, resources, and test directories.

2. **Live Reload**: The LiveReload feature automatically refreshes the browser upon redeployment.

   - `liveReload` Property: Enables or disables automatic browser refresh on redeployment. The `liveReload` property is set to `true` by default in the `dev` goal and `false` in the `start` goal.
   - `browser` Property: The `browser` property offers seamless configuration. If not specified, the system prioritizes Chrome and Firefox, gracefully falling back to Edge on Windows, Safari on macOS, or Firefox on other platforms. This improvement simplifies setup - no need to worry about specifying a browser; it adapts based on your system.
   - Cached Last Accessed URL: The last accessed Payara Server URL is stored in the `payara-maven-config.properties` file within the system's temporary directory. This ensures accessibility across application restarts, and eliminating the need for users to manually navigate back to the last accessed URL. This streamlined approach enhances the overall user experience by maintaining continuity and reducing manual intervention after each restart.

3. **Persistent Session State**: The `keepState` property allows for the persistence of

session state across multiple re-deployments during the development process. Enabled by default in the `dev` goal and disabled in the `start` goal.

4. **Readable Logging**: The `trimLog` property refactors the log format for improved readability. Enabled by default in the `dev` goal and disabled in the `start` goal.

> **NOTE** All listed features are enabled by default in the `dev` mode.

Supported Keywords

The Payara Server Maven plugin provides a built-in CLI interface that accepts specific keywords to interact with a running Payara Server instance. These commands can be used via terminal input.

- `asadmin <command>` Executes the specified `asadmin` subcommand. This is only available when the server manager is an instance of a local server.

```
Example:
[source,shell]
----
asadmin list-applications
----
```

- `deploy` Deploys the current application to the running server instance.
- `undeploy` Undeploys the current application from the running server instance.
- `exit` Exits the CLI session and terminates the running Payara server process.
- AI Agent Query

```
If the AI agent is enabled in the plugin configuration, the CLI accepts **any
natural language query**.
The query will be interpreted by the AI and responded to accordingly, based on
the current context.
```

> **NOTE** The predefined keywords (`asadmin`, `deploy`, `undeploy`, `exit`) will take precedence over AI interpretation if matched.

AI Agent (Experimental)

The AI Agent is an experimental feature in the Payara Server Maven plugin that brings intelligent automation and observability to Jakarta EE application development. It is enabled by default in `dev` mode and disabled in other modes for safety. To enable it manually, set the `payara.ai.agent` system property or the `PAYARA_AI_AGENT` environment variable to `true`.

Capabilities

The AI Agent enhances the developer experience by interpreting natural language instructions and extracting contextual insights from the running server. Key features include:

- Generating and explaining **`asadmin` commands** from plain English instructions.
- Querying server configuration and runtime state via the **Domain REST endpoints** and **JMX MBeans**.
- Executing **safe, read-only `asadmin` commands**, such as listing deployed applications or viewing JDBC connection pool status.
- Parsing and summarizing **server logs** and key configuration files like `domain.xml` to surface relevant insights.
- Offering intelligent prompts and auto-completions for common server administration and troubleshooting tasks.

Configuration Properties

The AI Agent can be customized using the following properties:

Property Name	System Property	Environment Variable	Default Value	Description
API Key	payara.ai.api.key	PAYARA_AI_API_KEY	*none*	API Key for accessing the AI provider's services.

Property Name	System Property	Environment Variable	Default Value	Description
Provider Location	payara.ai.provider.location	PAYARA_AI_PROVIDER_LOCATION	*none*	Endpoint or URL of the AI provider. For example, for `OLLAMA`, the location is `http://localhost:11434`.
Provider	payara.ai.provider	PAYARA_AI_PROVIDER	`OPEN_AI`	Name of the AI service provider. Supported values: `OPEN_AI`, `CUSTOM_OPEN_AI`, `GOOGLE`, `DEEPINFRA`, `DEEPSEEK`, `GROQ`, `MISTRAL`, `OLLAMA`, `ANTHROPIC`, `LM_STUDIO`, `GPT4ALL`.
Model	payara.ai.model	PAYARA_AI_MODEL	`gpt-4o-mini`	The AI model to be used (e.g., `gpt-4`, `claude-3-opus`, etc.).
Temperature	payara.ai.temperature	PAYARA_AI_TEMPERATURE	*none*	Controls the randomness of AI responses. Range: 0.0 (deterministic) to 1.0 (creative).
Top-P	payara.ai.top.p	PAYARA_AI_TOP_P	*none*	Controls diversity via nucleus sampling. An alternative to temperature.
Stream	payara.ai.stream	PAYARA_AI_STREAM	*none*	Enable streaming responses if supported by the provider.
Timeout	payara.ai.timeout	PAYARA_AI_TIMEOUT	*none*	Timeout in milliseconds for requests made to the AI provider.
Log Requests	payara.ai.log.requests	PAYARA_AI_LOG_REQUESTS	*none*	Log outgoing AI requests.

Property Name	System Property	Environment Variable	Default Value	Description
Log Responses	payara.ai.log.responses	PAYARA_AI_LOG_RESPONSES	*none*	Log incoming AI responses.
Repeat Penalty	payara.ai.repeatPenalty	PAYARA_AI_REPEAT_PENALTY	*none*	Controls penalty for repeated tokens in generated content.
Organization ID	payara.ai.organizationId	PAYARA_AI_ORGANIZATION_ID	*none*	AI provider organization ID, if required (e.g., OpenAI).
Top-K	payara.ai.topK	PAYARA_AI_TOP_K	*none*	Limit the next token sampling to the top-K most likely tokens.
Max Tokens	payara.ai.maxTokens	PAYARA_AI_MAX_TOKENS	*none*	Maximum number of tokens for a request.
Max Completion Tokens	payara.ai.maxCompletionTokens	PAYARA_AI_MAX_COMPLETION_TOKENS	*none*	Maximum number of tokens to generate in a completion.
Max Output Tokens	payara.ai.maxOutputTokens	PAYARA_AI_MAX_OUTPUT_TOKENS	*none*	Maximum number of tokens expected in AI response.
Presence Penalty	payara.ai.presencePenalty	PAYARA_AI_PRESENCE_PENALTY	*none*	Penalty for including new topics in completions.
Frequency Penalty	payara.ai.frequencyPenalty	PAYARA_AI_FREQUENCY_PENALTY	*none*	Penalty for repeating existing words/tokens in the input.
Custom Headers	payara.ai.customHeaders	PAYARA_AI_CUSTOM_HEADERS	*none*	Comma-separated custom headers to include in requests.
Seed	payara.ai.seed	PAYARA_AI_SEED	*none*	Seed value for deterministic output from the AI provider.
Allow Code Execution	payara.ai.allowCodeExecution	PAYARA_AI_ALLOW_CODE_EXECUTION	*none*	Allow executing code snippets in the AI responses.

Property Name	System Property	Environment Variable	Default Value	Description
Include Code Execution Output	payara.ai.includeCodeExecutionOutput	PAYARA_AI_INCLUDE_CODE_EXECUTION_OUTPUT	*none*	Include the result of code execution in AI output.
Max Retries	payara.ai.maxRetries	PAYARA_AI_MAX_RETRIES	*none*	Maximum number of retries on failure when querying the AI provider.
Chat History	payara.ai.chat.history	PAYARA_AI_CHAT_HISTORY	false	Enable context history in chat-based AI requests.
Chat History Limit	payara.ai.chat.history.limit	PAYARA_AI_CHAT_HISTORY_LIMIT	3	Number of previous interactions retained in chat history.

Usage Context

The AI Agent operates best during development, local testing, or in isolated environments. It is **not recommended for use in production** due to its access to sensitive runtime data and its experimental nature. By integrating AI-driven insights into the Payara tooling, the AI Agent aims to streamline operations, accelerate diagnostics, and boost developer productivity.

> **NOTE** As this is an experimental feature, usage and API compatibility may change in future versions.

Payara Micro Maven Plugin

Payara Micro has a Maven plugin which can start and stop Payara Micro and create an Uber JAR bundle of the application and Payara Micro.

The plugin requires JDK 8 or higher and the latest version is currently `1.4.0`

Configurable Goals

`bundle`

Aim: This goal bundles the attached project's artifact into an Uber JAR with the

specified configurations. By default, this goal is attached to the install phase. **Usage**: `mvn payara-micro:bundle`

Example:

```
<plugin>
    <groupId>fish.payara.maven.plugins</groupId>
    <artifactId>payara-micro-maven-plugin</artifactId>
    <version>${payara.micro.maven.plugin.version}</version>
    <executions>
        <execution>
            <goals>
                <goal>bundle</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <startClass>my.custom.start.class.Main</startClass>
        <deployArtifacts>
            <artifactItem>
                <groupId>org.mycompany</groupId>
                <artifactId>my-project</artifactId>
                <version>1.0</version>
                <type>ear</type>
            </artifactItem>
        </deployArtifacts>
        <customJars>
            <artifactItem>
                <groupId>org.primefaces</groupId>
                <artifactId>primefaces</artifactId>
                <version>6.0</version>
            </artifactItem>
        </customJars>
    </configuration>
</plugin>
```

Table 205. Configuration tags

Element	System Property	Environment Variable	Default	Notes
`autoDeployArtifact`	payara.auto.deploy.artifact	PAYARA_AUTO_DEPLOY_ARTIFACT	true	If the extension of the produced artifact is `WAR`, it will be copied automatically to the `MICRO-INF/deploy` directory when this property is set to `true`.

Element	System Property	Environment Variable	Default	Notes
`autoDeployContextRoot`	payara.auto.deploy.context.root	PAYARA_AUTO_DEPLOY_CONTEXT_ROOT	*none*	Sets context root of the deployed artifact if `autoDeployArtifact` is `true`. `autoDeployContextRoot` has the higher precedence compared to the `autoDeployEmptyContextRoot`.
`autoDeployEmptyContextRoot`	payara.auto.deploy.empty.context.root	PAYARA_AUTO_DEPLOY_EMPTY_CONTEXT_ROOT	`true`	Sets context root of the deployed artifact to `/` if `autoDeployArtifact` is `true`. If set to false, context root is derived from `project.build.finalName`. If `autoDeployContextRoot` is specified, `autoDeployEmptyContextRoot` will be ignored.
`startClass`	payara.start.class	PAYARA_START_CLASS	(optional):	Replaces the `Start-Class` definition that resides in the `MANIFEST.MF` file with the provided class.
`appendSystemProperties`	payara.append.system.properties	PAYARA_APPEND_SYSTEM_PROPERTIES	`true`	Appends all system properties defined into the `payara-boot.properties` file.

Element	System Property	Environment Variable	Default	Notes
`payaraMicroVersion`	payara.micro.version	PAYARA_MICRO_VERSION	`7.2.0`	By default, bundle mojo fetches Payara Micro with version 7.2.0. Note: `payaraVersion` is deprecated and has been replaced by `payaraMicroVersion` to stay in sync with the Payara Server Maven Plugin.
`deployArtifacts`			*None*	Can contain a list of artifactItems, which defines the dependencies with their GAVs to be copied under MICRO-INF/deploy folder.
`customJars`			*None*	Can contain a list of artifactItems, which defines the dependencies with their GAVs to be copied under MICRO-INF/lib folder.
`uberJarClassifier`	payara.uber.jar.classifier	PAYARA_UBER_JAR_CLASSIFIER	`microbundle`	Specifies the maven artifact classifier to use for the generated uber-jar.

The `bundle` goal always searches for domain configuration files in `src/main/resources` to copy in to the `MICRO-INF/domain` directory of the Uber jar. The files which are currently included are:

- `domain.xml`
- `hazelcast-config.xml`
- `keystore.jks`
- `login.conf`

- `logging.properties`
- `metrics.xml`

Additionally, boot command scripts will be included if they exist in `src/main/resources` and transferred to the `MICRO-INF` directory of the Uber jar artifact.

`pre-boot-commands.txt`: Provides a file of asadmin commands to run before booting the server. **`post-boot-commands.txt`**: Provides a file of asadmin commands to run after booting the server. **`post-deploy-commands.txt`**: Provides a file of asadmin commands to run after all deployments have completed.

`start`
Aim: This goal starts Payara Micro with the specified configuration. **Usage**: `mvn payara-micro:start`

Example:

```
<plugin>
    <groupId>fish.payara.maven.plugins</groupId>
    <artifactId>payara-micro-maven-plugin</artifactId>
    <version>${payara.micro.maven.plugin.version}</version>
    <executions>
        <execution>
            <goals>
                <goal>start</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <useUberJar>true</useUberJar>
        <payaraMicroAbsolutePath>/path/to/payara-
micro.jar</payaraMicroAbsolutePath>
        <payaraVersion>{release-id}</payaraVersion>
        <artifactItem>
            <groupId>fish.payara.extras</groupId>
            <artifactId>payara-micro</artifactId>
            <version>{release-id}</version>
        </artifactItem>
        <daemon>true</daemon>
        <javaPath>/path/to/Java/Executable</javaPath>
        <deployWar>true</deployWar>
        <contextRoot>myApp</contextRoot>
        <javaCommandLineOptions>
            <option>
                <value>-Xdebug</value>
            </option>
```

```
            <option>
                <key>-Xrunjdwp:transport</key>
                <value>dt_socket,server=y,suspend=y,address=5005</value>
            </option>
        </javaCommandLineOptions>
        <commandLineOptions>
            <option>
                <key>--domainconfig</key>
                <value>/path/to/domain.xml</value>
            </option>
            <option>
                <key>--autoBindHttp</key>
                <value>true</value>
            </option>
        </commandLineOptions>
    </configuration>
</plugin>
```

NOTE

If you want to execute the payara-micro plugin along with maven-toolchains-plugin, you need to execute the plugin as: `mvn toolchains:toolchain payara-micro:start`. See Using Toolchains for more information.

Table 206. Configuration tags

Element	System Property	Environment Variable	Default	Notes
`useUberJar`	payara.use.uber.jar	PAYARA_USE_UBER_JAR	false	Use the created uber-jar that resides in the target folder. The name of the JAR artifact will be resolved automatically by evaluating its final name, artifact ID and version. This configuration element has the higher precedence compared to `payaraMicroAbsolutePath`, `payaraVersion` and `artifactItem`.
`uberJarClassifier`	payara.uber.jar.classifier	PAYARA_UBER_JAR_CLASSIFIER	`microbundle`	Specifies the maven artifact classifier to use for the Uber JAR.

Element	System Property	Environment Variable	Default	Notes
`payaraMicroPath`	payara.micro.path	PAYARA_MICRO_PATH	*none*	Absolute path to the Payara Micro executable. Note: Parameter `payaraMicroAbsolutePath` is deprecated and has been replaced by `payaraMicroPath` to stay in sync with the Payara Server Maven Plugin.
`payaraMicroVersion`	payara.micro.version	PAYARA_MICRO_VERSION	`7.2.0`	By default, bundle mojo fetches Payara Micro with version 7.2.0. Note: `payaraVersion` is deprecated and has been replaced by `payaraMicroVersion` to stay in sync with the Payara Server Maven Plugin.
`artifactItem`			*none*	Defines payara-micro artifact with its coordinates. Specified artifact should be available in local maven repository.
`daemon`	payara.daemon	PAYARA_DAEMON	false	Starts Payara Micro in separate JVM process and continues with the maven build.

Element	System Property	Environment Variable	Default	Notes
`immediateExit`	payara.immediate.exit	PAYARA_IMMEDIATE_EXIT	false	If Payara Micro is executed in daemon mode, the executor thread will wait for the ready message before shutting down its process. By setting `immediateExit` to `true` you can skip this and instantly interrupt the executor thread.
`javaHome`	payara.java.home	PAYARA_JAVA_HOME	`java`	Absolute path to the java executable. This has higher priority to the java executable identified via the Maven toolchain. Note: Parameter `javaPath` is deprecated and has been replaced by `javaHome` to stay in sync with the Payara Server Maven Plugin.
`deployWar`	payara.deploy.war	PAYARA_DEPLOY_WAR	false	If the attached project is of type WAR, it will automatically be deployed if `deployWar` is set to `true`.
`contextRoot`	payara.context.root	PAYARA_CONTEXT_ROOT	*none*	Defines the context root of an application.
`javaCommandLineOptions`			*none*	Defines a list of command line options that will be passed to `java` executable. Command line options can either be defined as key-value pairs or just as list of values. key-value pairs will be formatted as `key=value`.

Element	System Property	Environment Variable	Default	Notes
`commandLineOptions`			*none*	Defines a list of command line options that will be passed onto payara-micro.
`hotDeploy`	payara.hot.deploy	PAYARA_HOT_DEPLOY	false	Enables the Hot Deploy mode.
`autoDeploy`	payara.auto.deploy	PAYARA_AUTO_DEPLOY	false	Enables automatic compilation and deployment upon saving files.
`liveReload`	payara.live.reload	PAYARA_LIVE_RELOAD	false	Triggers a browser refresh for an up-to-date view after each redeployment.
`browser`	payara.browser	PAYARA_BROWSER	none	Specifies the browser for live reload. Adapted based on the system if not specified.
`keepState`	payara.keep.state	PAYARA_KEEP_STATE	false	Persists session state across multiple redeployments.
`trimLog`	payara.trim.log	PAYARA_TRIM_LOG	false	Refactors log format for improved readability.

`stop`

Aim: This goal stops Payara Micro with the specified configuration. By default, this goal tries to find the currently executing Payara Micro instance by checking the running JAR.

If an `artifactItem` is defined, it will take precedence for identifying currently running instances. If `processId` is defined, this takes the highest precedence and the given `processId` will immediately kill the executing Payara Micro instance.

Usage: `mvn payara-micro:stop`

Example:

```
<plugin>
```

```
    <groupId>fish.payara.maven.plugins</groupId>
    <artifactId>payara-micro-maven-plugin</artifactId>
    <version>${payara.micro.maven.plugin.version}</version>
    <executions>
        <execution>
            <goals>
                <goal>stop</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <processId>32333</processId>
        <artifactItem>
            <groupId>fish.payara.extras</groupId>
            <artifactId>payara-micro</artifactId>
            <version>{release-id}</version>
        </artifactItem>
    </configuration>
</plugin>
```

NOTE

If you want to execute the payara-micro plugin along with maven-toolchains-plugin, you need to execute the plugin as: `mvn toolchains:toolchain payara-micro:start`. See Using Toolchains for more information.

Table 207. Configuration tags

Element	System Property	Environment Variable	Default	Notes
`processId`	payara.process.id	PAYARA_PROCESS_ID	*none*	Process id of the running Payara Micro instance.
`artifactItem`			*none*	Defines payara-micro artifact with its coordinates. This information is used to identify the process id of the running Payara Micro instance.
`useUberJar`	payara.use.uber.jar	PAYARA_USE_UBER_JAR	`false`	Use the name of the created uber-jar that resides in target folder to identify the process id of the running Payara Micro instance.

Element	System Property	Environment Variable	Default	Notes
`uberJarClassifier`	payara.uber.jar.classifier	PAYARA_UBER_JAR_CLASSIFIER	`microbundle`	Specifies the maven artifact classifier of the generated uber-jar to identify the process id of the running Payara Micro instance.
`maxStopTimeoutMillis`	payara.max.stop.timeout.millis	PAYARA_MAX_STOP_TIMEOUT_MILLIS	5000	Defines the maximum timeout in milliseconds to wait for the process of Payara Micro instance to stop.

`reload`

Aim: This goal reloads the Payara Micro application with the specified configuration. By default, this goal performs the vanilla deployment of the application without restarting the Payara Micro instance.

NOTE If `hotDeploy` is set to true then the Hot Deploy functionality will be enabled.

Usage: `mvn payara-micro:reload`

NOTE This goal is specifically designed for integration with the IDEs and Editors. Payara Micro instance allows IDEs to deploy the application in Hot Deploy mode by reusing the existing application instance and updating its classloader and internal components relative to the modified source.
This can lead to significant savings of time when an application is redeployed and boost developer productivity even more.

Table 208. Configuration tags

Element	System Property	Environment Variable	Default	Notes
`hotDeploy`	payara.hot.deploy	PAYARA_HOT_DEPLOY	false	Enables the Hot Deploy mode.
`sourcesChanged`	payara.sources.changed	PAYARA_SOURCES_CHANGED	*none*	Defines a list (comma separated) of the absolute paths to the source file which need to be reloaded via the Hot Deploy feature.

Element	System Property	Environment Variable	Default	Notes
`metadataChanged`	payara.metadata.changed	PAYARA_METADATA_CHANGED	false	If set to true, reloads deployment descriptors metadata in Hot Deploy mode
`keepState`	payara.keep.state	PAYARA_KEEP_STATE	false	Persists session state across multiple re-deployments.

`dev`

Aim: The dev goal is designed to streamline development by enabling `deployWar`, `exploded`, `autoDeploy`, `liveReload`, `keepState`, and `trimLog` to true, which facilitating auto compilation and deployment on saving files resulting in rapid compilation, and deployment for an efficient development workflow.

Usage:

```
mvn payara-micro:dev
```

The `dev` goal is an extended goal of `start` and is pre-configured for developing web applications in development mode which is equivalent to starting the Payara Micro instance using the following command:

```
mvn payara-micro:start -DautoDeploy=true -DliveReload=true -DdeployWar=true
-Dexploded=true -DkeepState -DtrimLog=true
```

Features in Dev Mode

1. **Auto Deploy**: The AutoDeploy feature enables automatic compilation and deployment of the application upon saving files within the project structure. Enabled by setting `autoDeploy=true`.

   The AutoDeploy feature relies on Java's WatchService to monitor changes in source directories, enabling continuous build and reload using Maven's Invoker. It dynamically modifies Maven goals based on file modifications, supports asynchronous execution for concurrent build tasks, and includes specifics for tracking changes in source, resources, and test directories.

WARNING	The AutoDeploy feature has a limitation and is not supported for `pom.xml`, `pre-boot-commands.txt`, `post-boot-commands.txt`, or `post-deploy-commands.txt`.

2. **Restart on modifying boot command scripts**: The restart functionality in `dev` mode enables a seamless instance restart upon modifications to the boot command scripts `pre-boot-commands.txt`, `post-boot-commands.txt`, or `post-deploy-commands.txt`, offering a quick server restart option for improved development efficiency.

WARNING	Modifying the `pom.xml` is not supported in this context, as limitations may arise when restarting with a modified `pom.xml`, potentially conflicting with previously passed properties via the

> command line.

3. **Live Reload**: The LiveReload feature automatically refreshes the browser upon redeployment.
    - `liveReload` Property: Enables or disables automatic browser refresh on redeployment. The `liveReload` property is set to `true` by default in the `dev` goal and `false` in the `start` goal.
    - `browser` Property: The `browser` property offers seamless configuration. If not specified, the system prioritizes Chrome and Firefox, gracefully falling back to Edge for Windows, Safari for macOS, or Firefox for other platforms. This improvement simplifies setup - no need to worry about specifying a browser; it adapts based on your system.
    - Cached Last Accessed URL: The last accessed Payara Micro URL is stored in the `payara-maven-config.properties` file within the system's temporary directory. This ensures accessibility across application restarts, eliminating the need for users to manually navigate to the last accessed URL. This streamlined approach enhances the overall user experience by maintaining continuity and reducing manual intervention after each restart.
4. **Persistent Session State**: The `keepState` property allows for the persistence of session state across multiple re-deployments during the development process. Enabled by default in the `dev` goal and disabled in the `start` goal.
5. **Readable Logging**: The `trimLog` property refactors the log format for improved readability. Enabled by default in the `dev` goal and disabled in the `start` goal.

> **NOTE** | All listed features are enabled by default in the `dev` mode.

Using Toolchains

The Maven Toolchains provide a way for plugins to discover what JDK (or other tools) are to be used during the build and Payara Micro Maven Plugin also supports using toolchains with its configuration.

In order to use the toolchains with either `payara-micro:start` or `payara-micro:stop`, first toolchains plugin should be defined as:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-toolchains-plugin</artifactId>
    <version>1.1</version>
    <executions>
        <execution>
            <goals>
                <goal>toolchain</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <toolchains>
            <jdk>
                <version>1.8</version>
                <vendor>oracle</vendor>
            </jdk>
        </toolchains>
    </configuration>
</plugin>
```

`toolchains.xml` is the file for configuring each toolchain, and it should reside under the `.m2` folder. A sample would be as:

```
<?xml version="1.0" encoding="UTF8"?>
<toolchains>
    <toolchain>
        <type>jdk</type>
        <provides>
            <version>1.7</version>
            <vendor>oracle</vendor>
        </provides>
        <configuration>

<jdkHome>/Library/Java/JavaVirtualMachines/jdk1.7.0_80.jdk/Contents/Home</jdkHome>
        </configuration>
    </toolchain>
    <toolchain>
        <type>jdk</type>
        <provides>
            <version>1.8</version>
            <vendor>oracle</vendor>
        </provides>
        <configuration>

<jdkHome>/Library/Java/JavaVirtualMachines/jdk1.8.0_131.jdk/Contents/Home</jdkHome>
        </configuration>
    </toolchain>
</toolchains>
```

The version and the vendor defined in the plugins section should match one of the entries defined in the toolchains.xml file. After configuring the toolchain, the plugin can be executed with `start` and `stop` goals as:

```
mvn toolchains:toolchain payara-micro:start
mvn toolchains:toolchain payara-micro:stop
```

Payara Micro Maven Archetype

Payara Micro has a Maven archetype which can create a sample Payara Micro application quickly.

Create a Project

To generate a new project from the archetype, run the following command :

```
$ mvn archetype:generate -DarchetypeGroupId=fish.payara.maven.archetypes
-DarchetypeArtifactId=payara-micro-maven-archetype -DarchetypeVersion=1.0.1
-DgroupId=fish.payara.micro -DartifactId=micro-sample -Dversion=1.0-SNAPSHOT
-Dpackage=fish.payara.micro.sample -Darchetype.interactive=false
```

Maven will ask for the project detail confirmation. Press enter or press Y.

```
Confirm properties configuration:
groupId: fish.payara.micro
artifactId: micro-sample
version: 1.0-SNAPSHOT
package: fish.payara.micro.sample
autoBindHttp: true
javaeeVersion: 8.0
payaraMicroVersion: 5.182
 Y: :
```

Now Maven will start creating the project structure and will display the following :

```
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: payara-
micro-maven-archetype:1.0.1
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: fish.payara.micro
[INFO] Parameter: artifactId, Value: micro-sample
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: fish.payara.micro.sample
[INFO] Parameter: packageInPathFormat, Value: fish/payara/micro/sample
[INFO] Parameter: payaraMicroVersion, Value: 5.182
[INFO] Parameter: package, Value: fish.payara.micro.sample
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
```

```
[INFO] Parameter: javaeeVersion, Value: 8.0
[INFO] Parameter: groupId, Value: fish.payara.micro
[INFO] Parameter: artifactId, Value: micro-sample
[INFO] Parameter: autoBindHttp, Value: true
[INFO] Project created from Archetype in dir: D:\micro-sample
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9.067 s
[INFO] Final Memory: 15M/196M
[INFO] ------------------------------------------------------------------------
```

Run the Project

```
$ mvn package payara-micro:start
```

Payara Micro Gradle Plugin

Payara Micro has a Gradle plugin which can start and stop Payara Micro and create an Uber JAR bundle of the application and Payara Micro .

NOTE	The plugin requires JDK 8 or higher and the latest plugin version is currently `1.1.0`

Installation

To use it, simply add the following script to your `build.gradle`:

```
plugins {
  id "fish.payara.micro-gradle-plugin" version "1.1.0"
}
```

Configuration Example

```
payaraMicro {
    payaraVersion = '7.2.0'
    deployWar = false
    useUberJar = true
    daemon = false
    commandLineOptions = [port: 2468]
    javaCommandLineOptions = [Dtest: 'test123', ea:true]
}
```

Plugin Tasks

`microBundle`

Aim: This task bundles the attached project's artifact into an Uber JAR with the

specified configurations. **Usage**: `gradle microBundle`

Table 209. Configuration tags

Element	Default	Notes
`javaPath`	java	Absolute path to the java executable
`payaraVersion`	7.2.0	By default `microBundle` task fetches payara-micro with version 7.2.0

`microStart`

Aim: This task starts Payara Micro with the specified configuration. **Usage**: `gradle microStart`

Table 210. Configuration tags

Element	Default	Notes
`useUberJar`	false	Use created uber-jar that resides in target folder. The name of the JAR artifact will be resolved automatically by evaluating its final name, artifact ID and version. This configuration element has the higher precedence compared to `payaraMicroAbsolutePath` and `payaraVersion`.
`payaraMicroAbsolutePath`	*none*	Absolute path to payara-micro executable.
`payaraVersion`	7.2.0	By default `microStart` task fetches Payara Micro with version 7.2.0.
`daemon`	false	Starts Payara Micro in separate JVM process and continues with the Gradle build.
`immediateExit`	false	If Payara Micro is executed in daemon mode, the executor thread will wait for the ready message before shutting down its process. By setting `immediateExit` to `true` you can skip this and instantly interrupt the executor thread.
`javaPath`	`java`	Absolute path to the java executable.
`deployWar`	false	If the attached project is of type WAR, it will automatically be deployed to Payara Micro if `deployWar` is set to `true`.

Element	Default	Notes
`javaCommandLineOptions`	*none*	Defines a list of command line options that will be passed to `java` executable. Command line options can either be defined as key-value pairs or just as list of values. key-value pairs will be formatted as `key=value`.
`commandLineOptions`	*none*	Defines a list of command line options that will be passed onto Payara Micro.
`hotDeploy`	false	Enables the hot reload mode.

`microStop`

Aim: This task stops Payara Micro with the specified configuration. By default this task tries to find out currently executing Payara Micro instance based on project GAV. If `processId` is defined, it will take precedence and the given `processId` will immediately kill the executing Payara Micro instance. **Usage**: `gradle microStop`

Table 211. Configuration tags

Element	Default	Notes
`processId`	*none*	Process id of the running Payara Micro instance.
`useUberJar`	false	Use created uber-jar that resides in build\libs folder. The name of the jar artifact will be resolved automatically by evaluating its final name.

`microReload`

Aim: This task reloads the Payara Micro application with the specified configuration. By default, this task performs the vanilla deployment of the application without restarting the Payara Micro instance. If `hotDeploy` is set to true then the Hot Reload functionality will be enabled. **Usage**: `gradle microReload`

IMPORTANT	This task is specifically designed for integration with the IDEs and Editors. Payara Micro instance allows IDEs to deploy the application in Hot Deploy mode by reusing the existing application instance and updating its classloader and internal components relative to the modified source. This can lead to significant

> savings of time when an application is redeployed and boost developer productivity even more.

Table 212. Configuration tags

Element	Default	Notes
`hotDeploy`	false	Enables the hot reload mode.
`sourcesChanged`	*none*	Defines a list(comma separated) of the absolute paths to the source file which need to be reloaded via the Hot Reload feature.
`metadataChanged`	false	If set to true, reloads deployment descriptors metadata in hot deploy mode

See Also

- Plugin page on Gradle Plugins Website

Maven Regex Profile Activation Extension

The Maven Regex Profile Activation extension allows activating profiles by checking that a property matches a given regular expression, rather than an exact value.

Usage

The Regex Profile Activator Maven Extension can be found on Maven Central and can be included in your project using the following Maven coordinates:

```
<dependency>
    <groupId>fish.payara.maven.extensions</groupId>
    <artifactId>regex-profile-activator</artifactId>
    <version>${release-id}</version>
</dependency>
```

You can find the latest artifact version from here.

Example

This extension is now active in your build, and you can activate profiles using a regex string beginning and ending with a `/`.

Example usage of Maven Regex Profile Activation in a `pom.xml` file

```
<profile>
    <id>test-profile</id>
    <activation>
        <property>
            <name>myproperty</name>
            <value>/abra.+/</value>
        </property>
    </activation>
</profile>
```

The profile above will be active when built with `-Dmyproperty=abracadabra`, or anything else is matching the regex.

In order to release the artifact you will need to sign the source, javadoc and artifact JARs. To do this, build the project with the `sign` profile:

```
mvn install -Dsign
```

Payara Starter Documentation

Introduction

Payara Starter is a web application and command line tool designed to accelerate Jakarta EE development by simplifying the creation of new application projects. It supports the creation of both Maven and Gradle projects, allowing developers to focus on coding their application components in a faster way as they don't have to deal with the complexities of setting up their projects.

- Website
- GitHub Repository
- Maven Central Repository for Payara Starter Archetype

Key Features of Payara Starter

- Build System Flexibility: Supports Maven and Gradle as Project Management tools for seamless integration with preferred build systems.
- Customizable Project Details: Define crucial project specifics upfront, aligning with organizational naming conventions.

- Java and Jakarta EE Version Control: Allows the selection of specific versions to ensure compatibility and leverage the latest features.
- MicroProfile Integration: Includes options to integrate MicroProfile APIs for building resilient, scalable services.
- Azul Payara Compatibility: Effortlessly works with Azul Payara, ensuring readiness for deployment.
- Docker Support: Facilitates Docker compatibility, streamlining containerization for agile deployment.
- Unit Test Templates: Optionally includes test templates, encouraging best testing practices from project initiation.
- Advanced Security Configuration: Optional advanced security setup for robust authentication via Jakarta Security, facilitating secure application development from the start.

Supported Jakarta EE Profiles

Payara Starter supports various Jakarta EE profiles:

- Platform: `jakarta.platform:jakarta.jakartaee-api`
- Web Profile: `jakarta.platform:jakarta.jakartaee-web-api`
- Core Profile: `jakarta.platform:jakarta.jakartaee-core-api`

Generating a New App Using Payara Starter

You can generate a new application using Payara Starter in two ways:

Command Line Using Archetype

To generate a new application using the Payara Starter archetype from the command line, use the following Maven command:

```
mvn archetype:generate -DarchetypeGroupId=fish.payara.starter -
DarchetypeArtifactId=payara-starter-archetype -DarchetypeVersion=1.0-beta4 <other
options>
```

This command generates a new application based on specified options. Adjust properties according to your project requirements.

Payara Starter Archetype Parameter

The Payara Starter Archetype has a robust set of parameters that allows the customization of various aspects during the application generation process. Below is a table detailing each parameter, its default value, and additional notes:

Element	Default Value	Notes
`build`	`maven`	Specifies the preferred build system for the project. Options include: `maven`, `gradle`.
`groupId`	`fish.payara`	Specifies the Group ID for the generated project.
`artifactId`	`hello-world`	Determines the Artifact ID for the generated project.
`version`	`0.1-SNAPSHOT`	Sets the version of the generated project.
`package`	`fish.payara`	Defines the base package name for the project.
`javaVersion`	`21`	Sets the Java version for the project. Available options: • 8 • 11 • 17 • 21 **WARNING** | Keep in mind that versions older than `11` are not supported in Azul Payara 7.2.0

Element	Default Value	Notes
`jakartaEEVersion`	`10`	Specifies the targeted Jakarta EE version. Valid options: • 8 • 9 • 9.1 • 10 **WARNING** | For compatibility with Azul Payara 7.2.0, versions older than `10` are not supported.
`profile`	`full`	Specifies the Jakarta EE profile options: 'core', 'web', or 'full'.
`platform`	`server`	Specifies the target distribution where the application will be deployed Options include: • `server` for Payara Server. • `micro` for Payara Micro.
`payaraVersion`	`7.2.0`	Sets the targeted Payara version. Retrieved from Maven Central.
`addConcurrentApi`	`false`	Includes support for the Concurrent API if `true`.
`addResourceApi`	`false`	Includes support for the Resource API if `true`.
`addJBatchApi`	`false`	Includes support for the JBatch API if `true`.
`addMicroprofileApi`	`true`	Includes support for the MicroProfile API if `true`.
`addJcache`	`false`	Includes support for the JCache if `true`.
`addPayaraApi`	`true`	Includes support for the Payara Public API if `true`.

Element	Default Value	Notes
`deployWar`	`true`	Automatically deploys the WAR file if `true`. **NOTE** | Applicable only for Payara Micro deployments.
`autoBindHttp`	`true`	Enables automatic HTTP binding. **NOTE** | Applicable only for Payara Micro deployments.
`contextRoot`	`/`	Sets the web context root for the deployed artifact. **NOTE** | Applicable only for Payara Micro deployments.
`includeTests`	`false`	Includes a set of generated Unit tests if `true`. **WARNING** | Currently limited to Maven projects.
`docker`	`false`	Enables Docker compatibility.
`mpConfig`	`false`	Enables support for the MicroProfile Config API.
`mpOpenAPI`	`false`	Enables support for MicroProfile OpenAPI.
`mpFaultTolerance`	`false`	Enables support for MicroProfile Fault Tolerance.
`mpMetrics`	`false`	Enables support for MicroProfile Metrics.
`auth`	`none`	Specifies the authentication type for the application. Available choices: • `none` • `formAuthFileRealm` • `formAuthDB` • `formAuthLDAP`

Web Application at `start.payara.fish`

Visit start.payara.fish to use the web application for generating applications via a graphical interface.

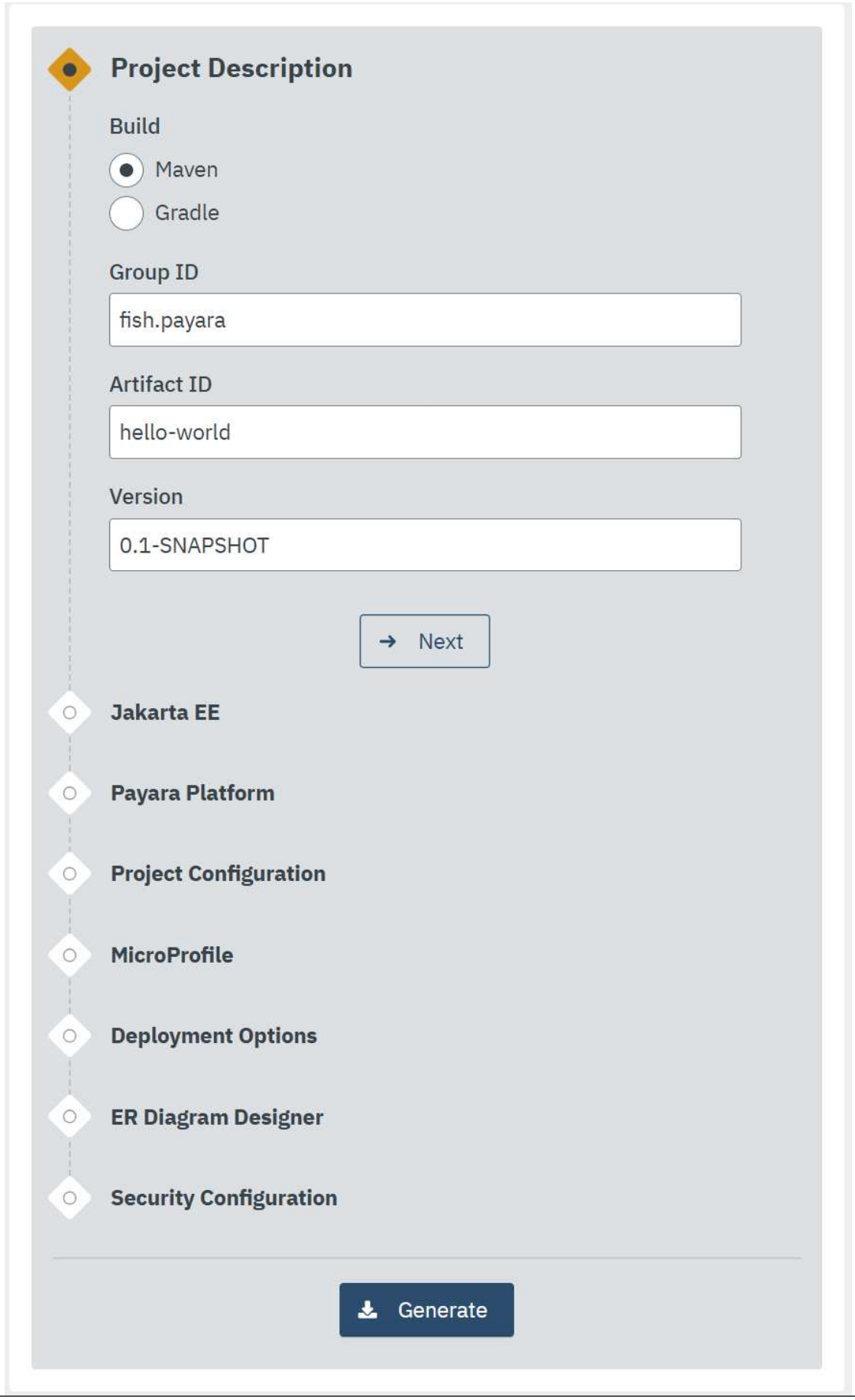
Using the Payara Starter Web Application

Payara Starter provides a convenient web application that allows you to generate Jakarta EE projects via a graphical interface.

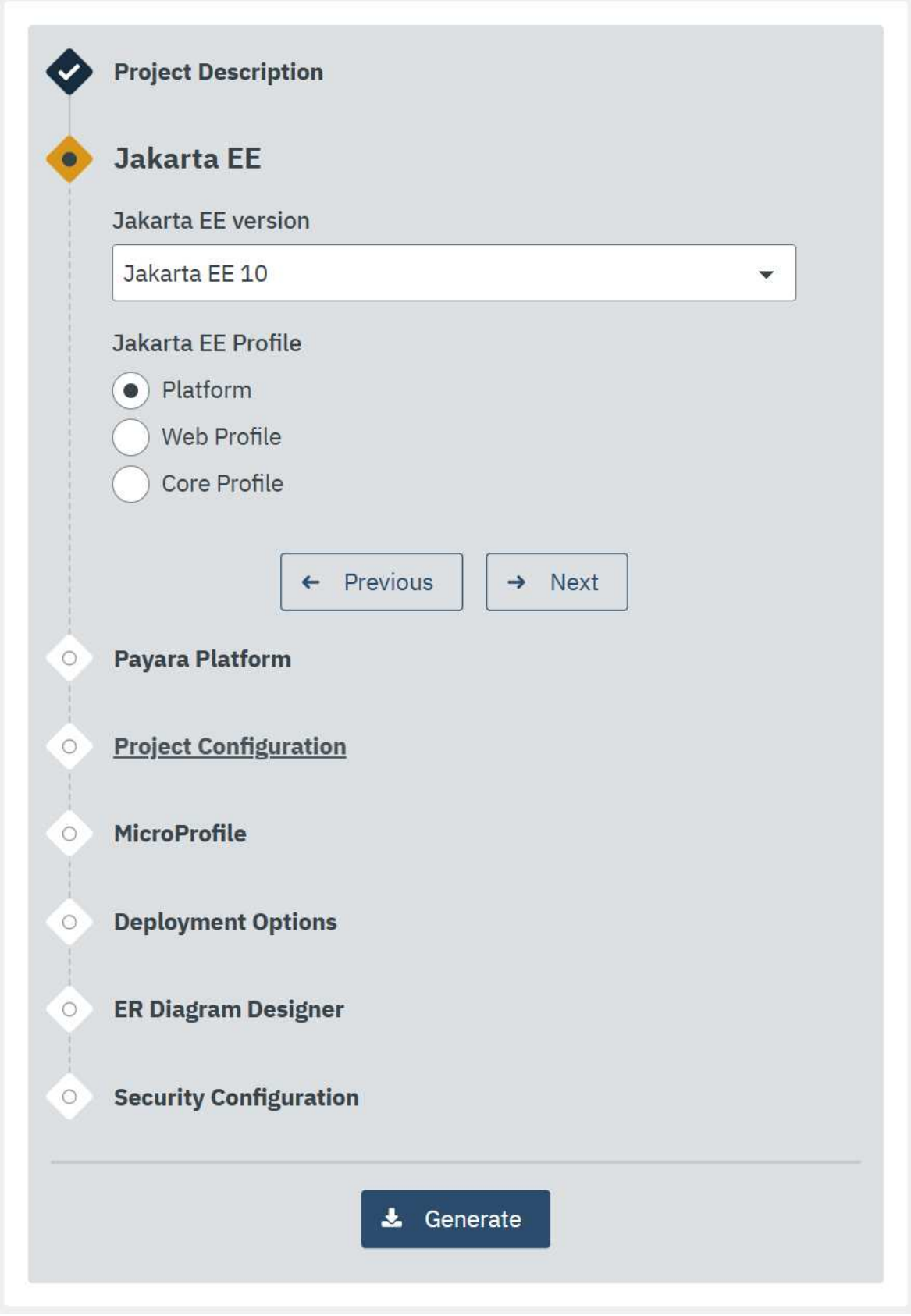
Follow these steps to use the web application:

1. Access the web application:
    - Visit the Payara Starter web application at https://start.payara.fish.
2. Fill in the **Project Details**:
    - On the web application's landing page, you'll find a user-friendly interface to input project details.
    - Enter the required information, such as **Group ID**, **Artifact ID**, and other relevant details.

      You can customize these based on your project requirements:

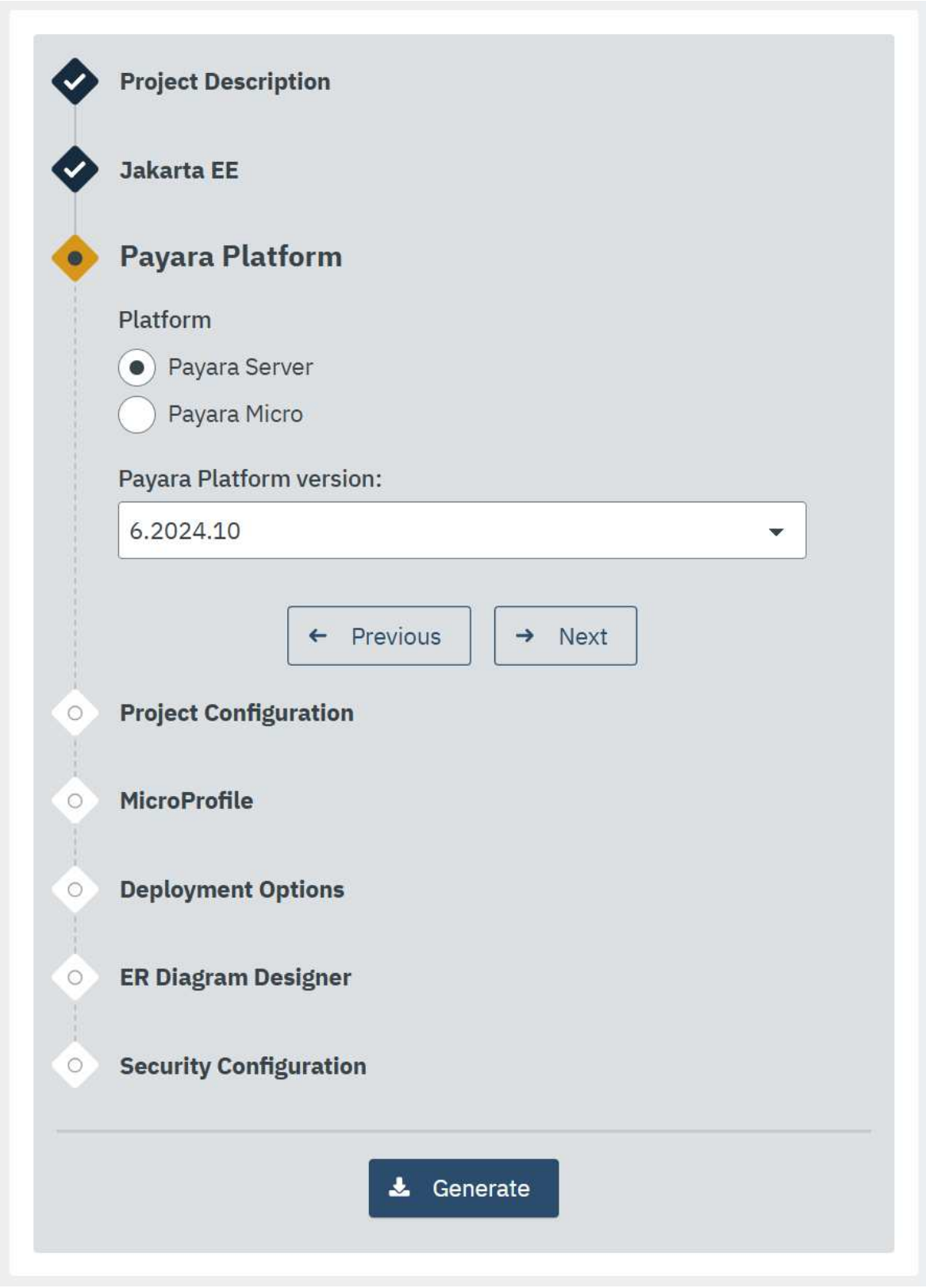

Project Description
Build
Maven
Gradle
Group ID
fish.payara
Artifact ID
hello-world
Version
0.1-SNAPSHOT
Next
Jakarta EE
Payara Platform
Project Configuration
MicroProfile
Deployment Options
ER Diagram Designer
Security Configuration
Generate

3. Choose the **Build System** and **Jakarta EE Version**:
    - Select your preferred build system (Maven or Gradle) from the provided options.
    - Choose the desired Jakarta EE version that aligns with your project’s compatibility requirements.



4. Configure any **Additional Options**:

- The web application allows you to configure additional options such as MicroProfile integration, Azul Payara version, and more.
- Customize these options based on your project needs.



5. Review and Generate:
    - Review the summary of your selected options to ensure they match your project

requirements.

- Click the **Generate** button to initiate the project generation process.

6. Download the Generated Project:
   - Once the generation process is complete, the web application triggers the download of the generated project archive.
   - Download the project archive, and you can then proceed to import it into your preferred Integrated Development Environment (IDE).

> **NOTE**
>
> The web application simplifies the project creation process but may not expose all configuration options available through the command-line archetype generation.
> If you need better control of how the application is generated, we recommend using the starter archetype instead.

Sample Generated Jakarta REST Resource

Here's an example of a generated Jakarta REST resource using Payara Starter:

```
package fish.payara.hello;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.Response;

@Path("hello")
public class HelloWorldResource {

    @GET
    public Response hello(@QueryParam("name") String name) {
        if ((name == null) || name.trim().isEmpty()) {
            name = "world";
        }
        return Response.ok(name).build();
    }
}
```

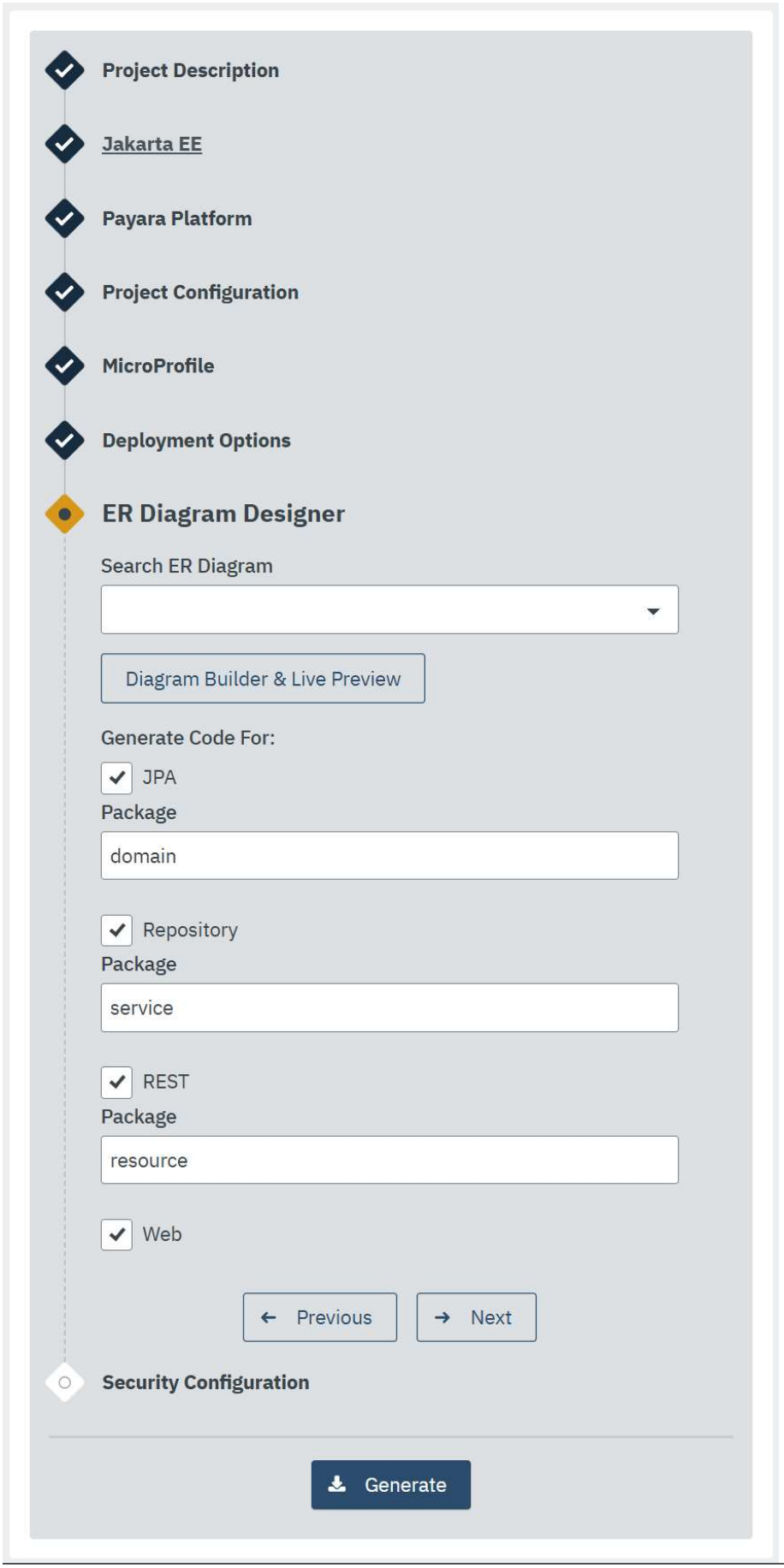
This sample demonstrates a simple REST endpoint named `hello` that accepts a query parameter `name` and responds with a greeting message.

Feel free to adjust archetype parameters to tailor the generated application to your

project requirements.

ER Diagram Designer with Generative AI Support

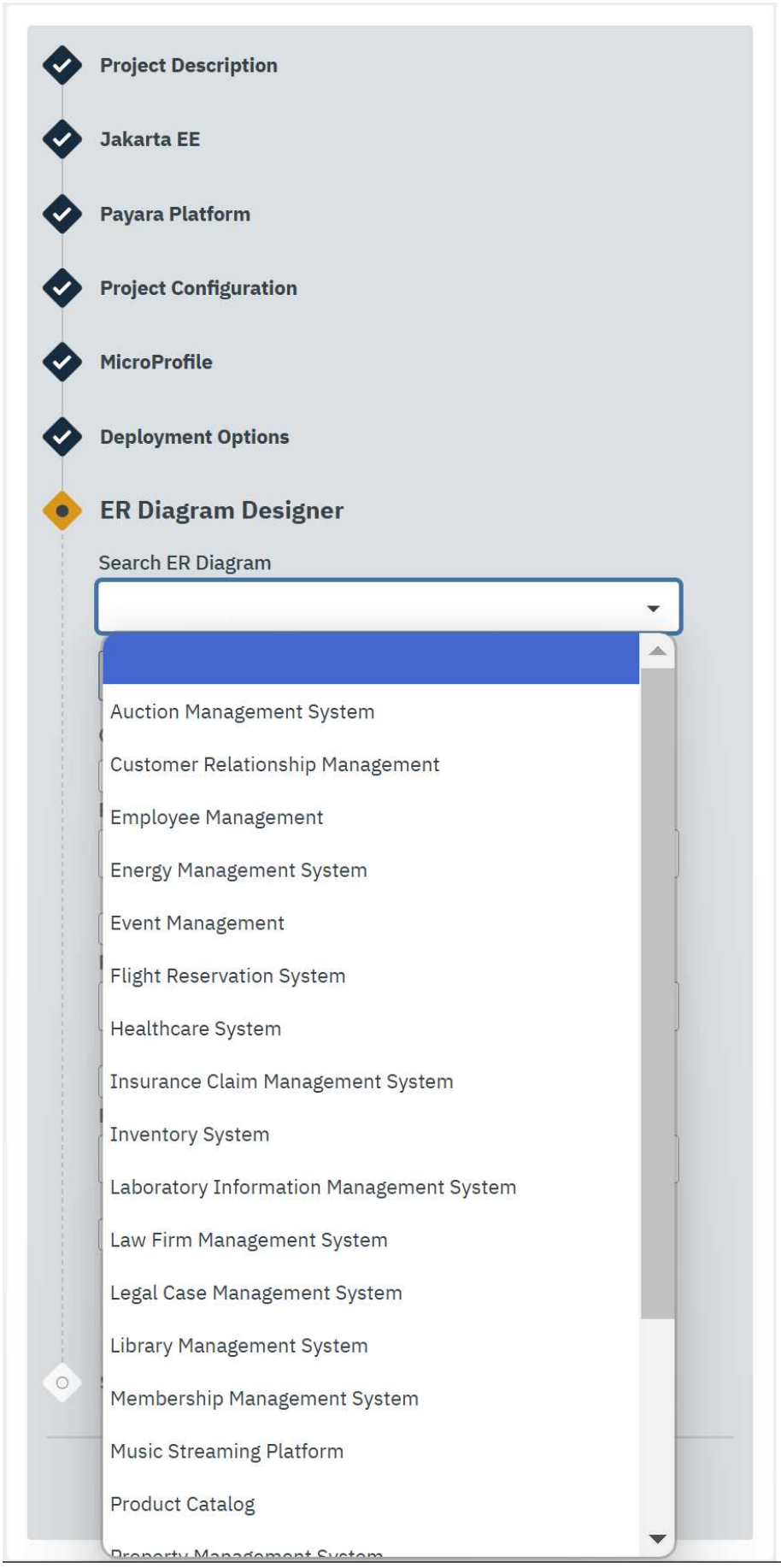
Payara Starter start.payara.fish enables developers to streamline the creation of full-stack applications by leveraging Entity-Relationship (ER) diagrams. Generative AI enhances this process by automating the generation and customization of diagrams based on inputs and requirements. Developers can visualize, interact, and modify these diagrams in real time, making Payara Starter a more powerful tool for application design and development.


Project Description
Jakarta EE
Payara Platform
Project Configuration
MicroProfile
Deployment Options
ER Diagram Designer
Search ER Diagram
Diagram Builder & Live Preview
Generate Code For:
JPA
Package
domain
Repository
Package
service
REST
Package
resource
Web
Previous
Next
Security Configuration
Generate

Features of ER Diagram Designer

Default ER Diagrams

The Payara Starter website includes default ER diagrams, stored in Mermaid format, which users can modify as a starting point for their own designs.


Project Description
Jakarta EE
Payara Platform
Project Configuration
MicroProfile
Deployment Options
ER Diagram Designer
Search ER Diagram
Auction Management System
Customer Relationship Management
Employee Management
Energy Management System
Event Management
Flight Reservation System
Healthcare System
Insurance Claim Management System
Inventory System
Laboratory Information Management System
Law Firm Management System
Legal Case Management System
Library Management System
Membership Management System
Music Streaming Platform
Product Catalog

Automated ER Diagram Generation

Payara Starter automates ER diagram generation by leveraging Generative AI based on user-provided descriptions. Users can create accurate and well-structured diagrams with minimal effort.

Real-Time Diagram Design via Free-Text Support

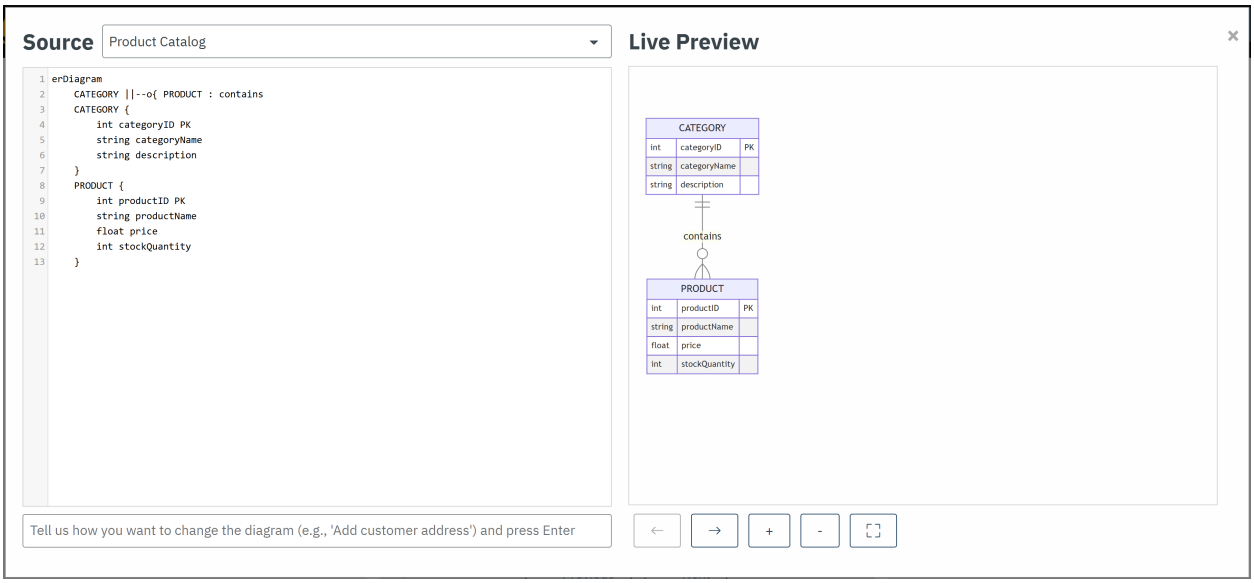
Through **Real-time Diagram Design via Free-text Support**, users can interact with Generative AI to refine and enlarge existing diagrams based on their specific requirements. This feature allows users to quickly modify or create new diagrams using free-form text commands.

WARNING	Generative AI is a helpful assistant for designing diagrams, but it is not perfect. Not all instructions will be understood as intended, and some results may differ from user expectations.

Instructions and Examples

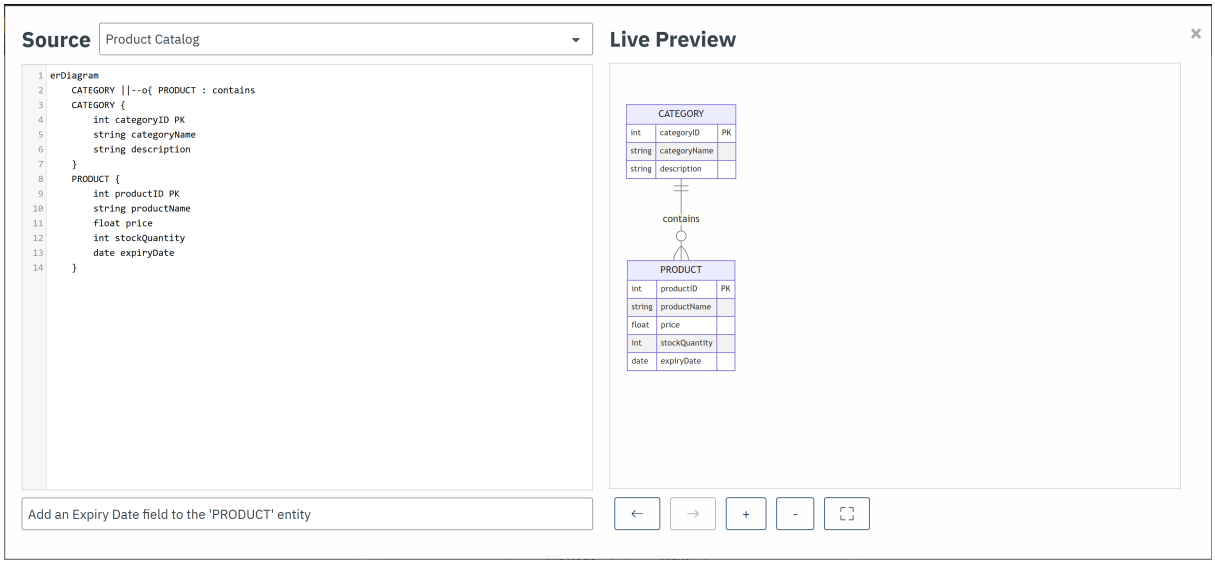
To get the best results, users should provide clear, concise commands. Below are a few sample scenarios to illustrate how users can modify their diagrams.

Starting from this default diagram:



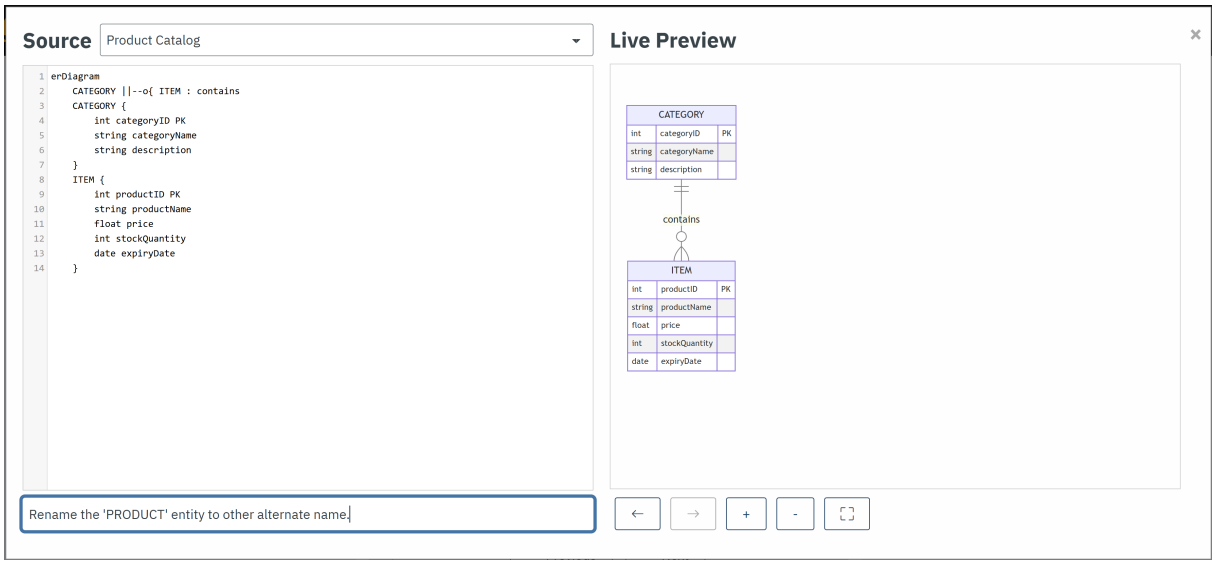
1. **Adding a new Field**: The following command prompts the AI to include a new attribute named `expiryDate` in the existing `PRODUCT` entity.

   Command: *Add an Expiry Date field to the `PRODUCT` entity.*



2. **Change an Entity Name**: The following command directs the AI to change the name of the `PRODUCT` entity, allowing for a new designation that may better reflect its purpose or context within the application.

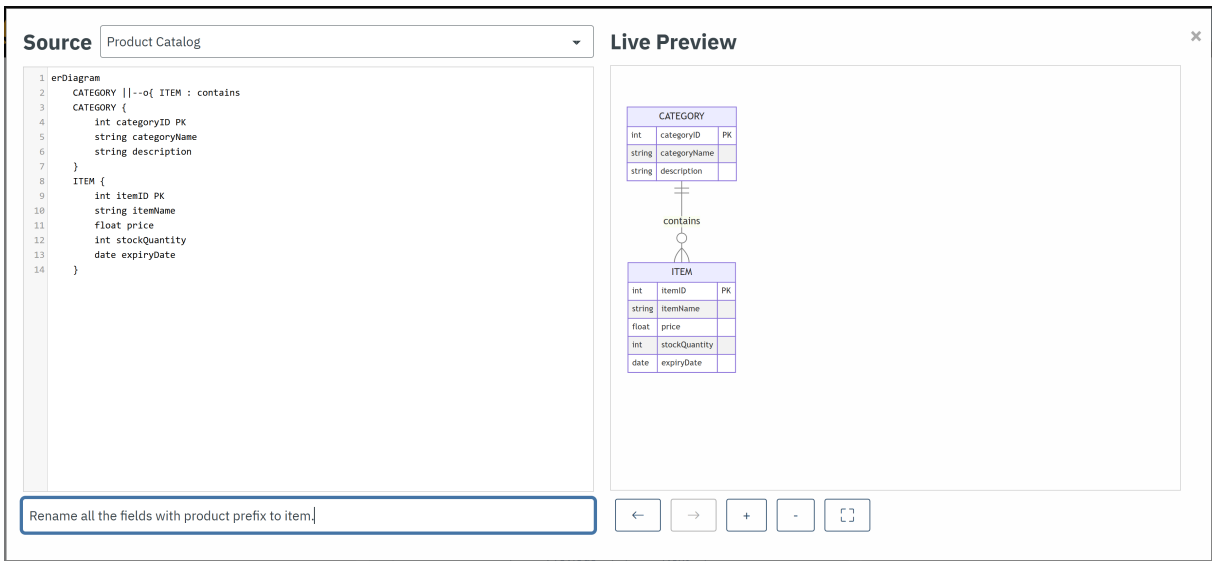
   Command: *Rename the `PRODUCT` entity to other alternate name.*



3. **Modify a Field**: The following command results in the AI updating the names of all fields that begin with `product`, replacing that prefix with `item`, thus standardizing or

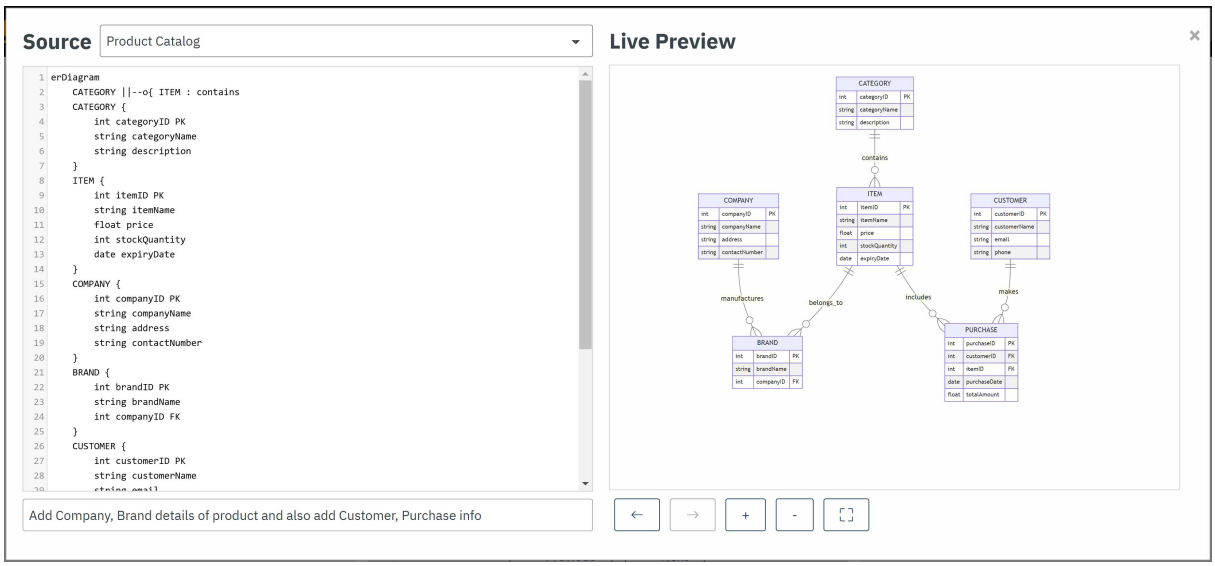
rebranding the fields for consistency.

Command: *Rename all the fields with* `product` *prefix to `item`.*



4. **Enhance the diagram by adding more entities and relationship**: The following command instructs the AI to enrich the diagram by introducing new entities (`Company`, `Brand`, `Customer`, `Purchase`) and defining relationships between them, thus expanding the data model to cover more aspects of the application domain.

   Command: *Add* `Company`, `Brand` *details of product and also add* `Customer`, `Purchase` *info.*



IMPORTANT	Users should review and modify the diagram after the AI-assisted prototype is generated. For more complex designs or detailed customizations, manual changes may be necessary to ensure the final entity relationship model meets all requirements. The AI is designed to speed up the process of generating diagram prototypes, but the user is responsible for creating the complete and accurate design of the application's business mode in its source code.

Interactive Auto-Pilot Mode

An innovative autopilot mode allows users to dynamically enlarge or shrink the number of entities in the ER diagram using (`+`/`–`) buttons, providing a quick way to modify complexity without detailed configuration.

*Enlarge (`)*: Adds additional entities or relationships to the diagram based on the current context. For example, clicking the ` button might introduce a related table, such as an `Order` entity linked to an existing `Customer` entity, along with predefined attributes or relationships.

Shrink (`–`): Removes entities or relationships from the diagram to simplify its structure. For instance, clicking the `–` button could remove auxiliary entities that are not critical to the application's initial design.

This feature helps users quickly prototype or refine their diagrams by adjusting the scope and complexity of the database structure, without requiring manual edits to every detail.

Live Diagram Panel

The Payara Starter interface includes a split panel where the Mermaid source code appears on the left, and a live visual representation of the ER diagram is shown on the right. Any changes made in the source or through Generative AI interactions update the visualization immediately.

Users can freely modify the ER diagram by editing the Mermaid source code directly in the left panel. These modifications are instantly reflected in the live preview, offering full control over the diagram's structure and appearance.

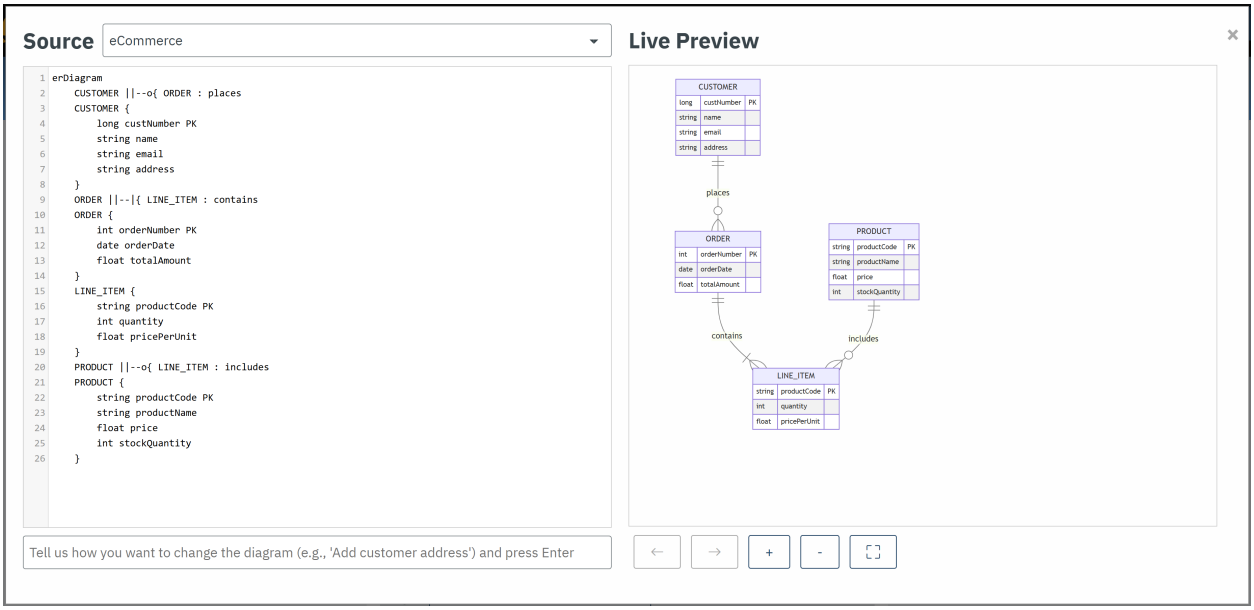
Full-Stack Application Generation

Once finalized, Payara Starter uses the ER diagram to automatically generate corresponding JPA entities and REST endpoints, making it easy to move from design to implementation.

Using the ER Diagram

Open the Existing or Generate a New ER Diagram

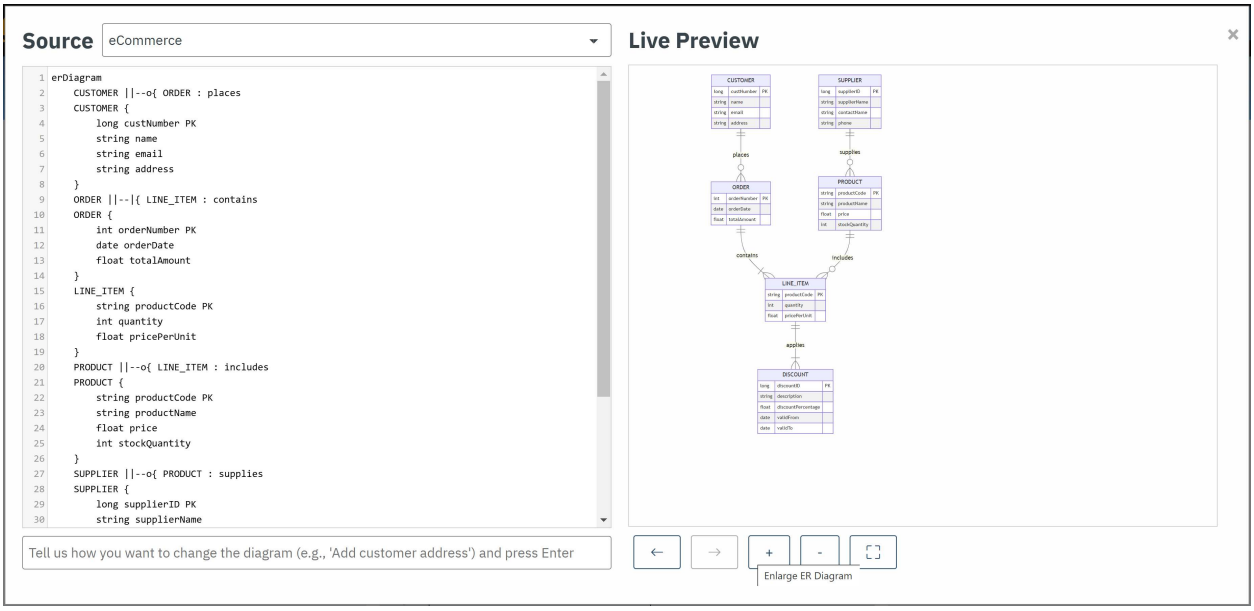
Open the ER Diagram panel within the Payara Starter application interface. Select from the default diagrams or start a new diagram by pressing the **Diagram Builder & Live Preview** button and entering a description.



Enhance the Diagrams

Use the Real-time Diagram Design via Free-text support to interact with OpenAI. Type specific requests or commands, such as *"Add an Order entity"* or *"Make Customer and Order related"* and receive real-time updates to your diagram.

In autopilot mode, use the (`+`/`−`) buttons to quickly enlarge or shrink the diagram.



Generate the Full-Stack Application

Once the ER diagram meets your requirements, you can initiate the full-stack application generation. Payara Starter uses the diagram structure to create JPA entities, repository services, REST endpoints, and optionally web pages.

Optional Features and Dependencies

During the generation process, you can customize the output to include or exclude specific components. Note the cascading dependencies between components:

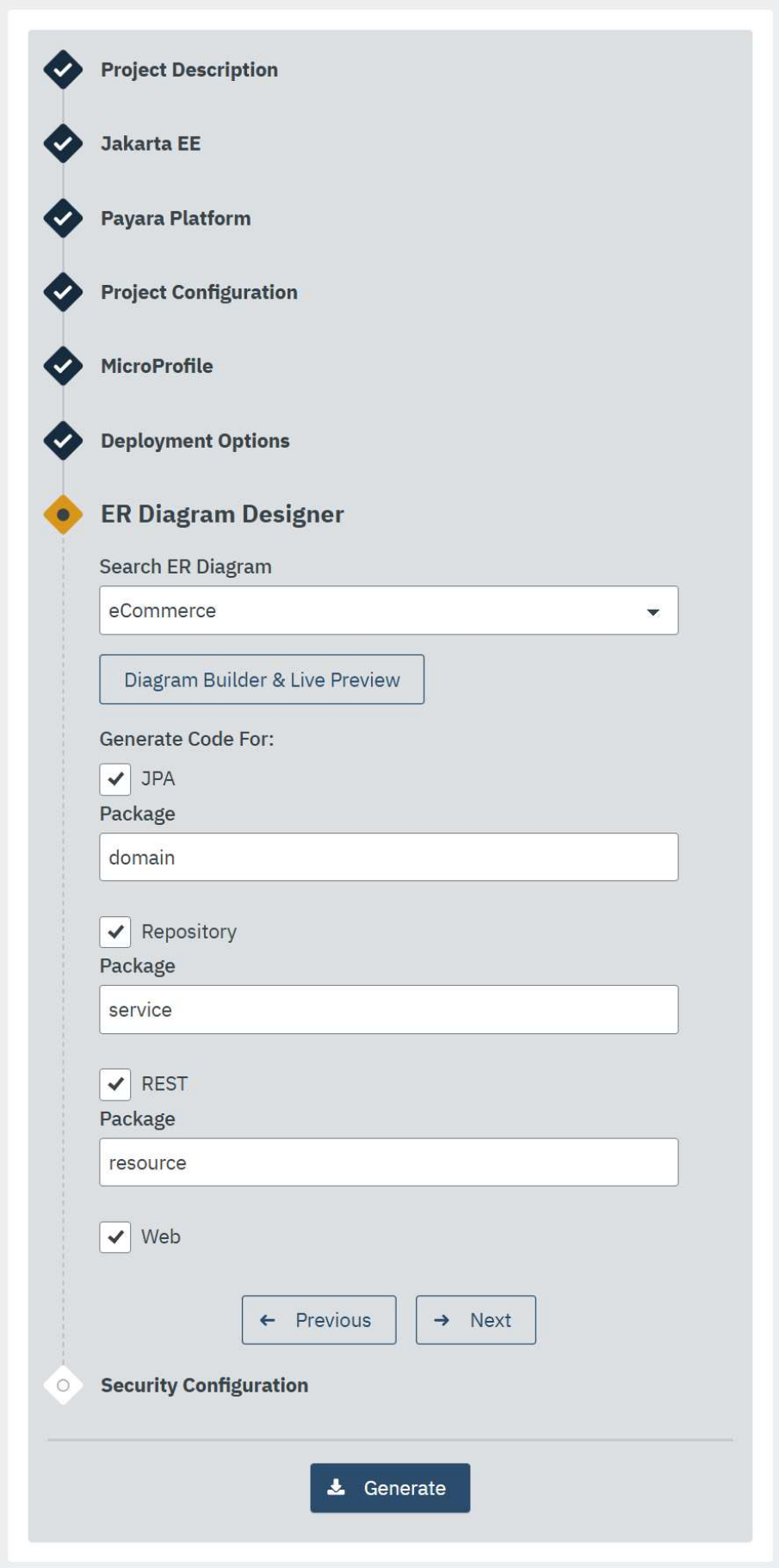
JPA Entities: If skipped, all other components (repository services, REST endpoints, and web pages) will also be skipped. Only the ER diagram will be saved to the root of the project.

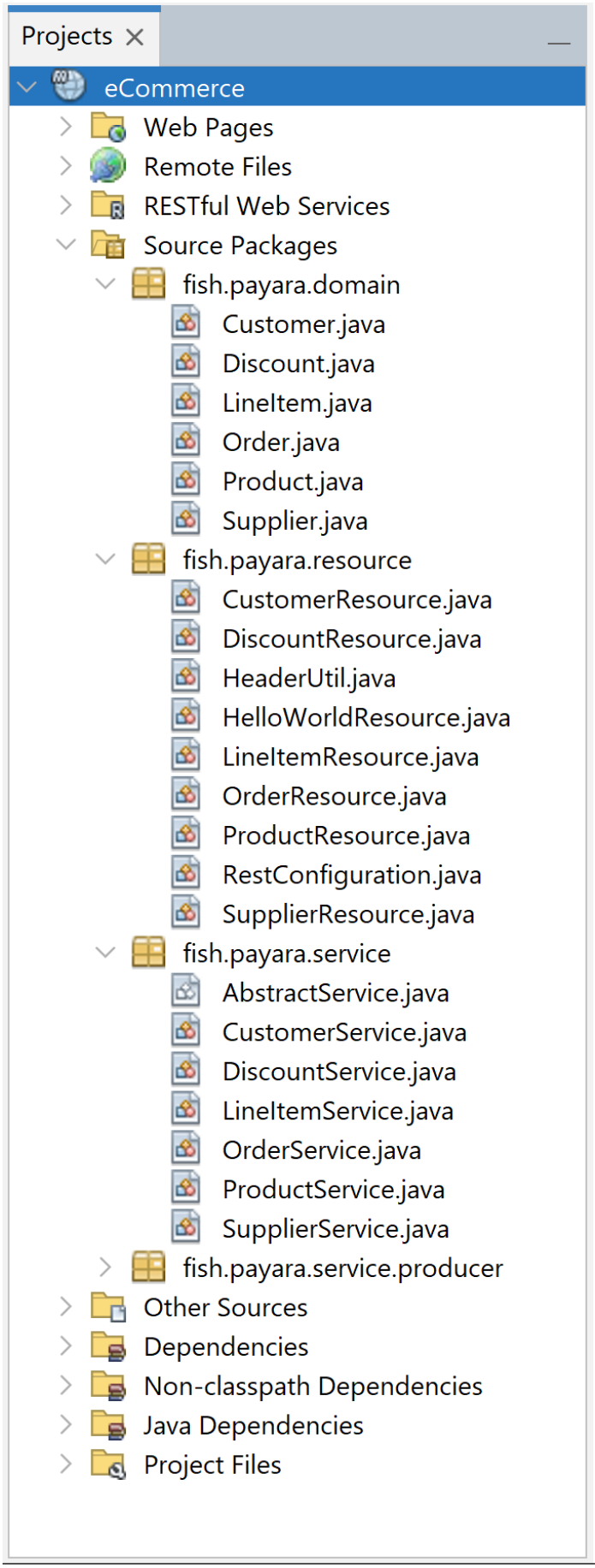
Repository Services: If skipped, the generation of REST endpoints and web pages will also be skipped.

REST Endpoints: If skipped, web pages will also be skipped.

Web Pages: Can only be generated if repository services and REST endpoints are included.

This flexibility allows developers to tailor the generated application to meet their specific project needs while ensuring structural coherence.


Project Description
Jakarta EE
Payara Platform
Project Configuration
MicroProfile
Deployment Options
ER Diagram Designer
Search ER Diagram
eCommerce
Diagram Builder & Live Preview
Generate Code For:
JPA
Package
domain
Repository
Package
service
REST
Package
resource
Web
Previous
Next
Security Configuration
Generate


Projects
eCommerce
Web Pages
Remote Files
RESTful Web Services
Source Packages
fish.payara.domain
Customer.java
Discount.java
LineItem.java
Order.java
Product.java
Supplier.java
fish.payara.resource
CustomerResource.java
DiscountResource.java
HeaderUtil.java
HelloWorldResource.java
LineItemResource.java
OrderResource.java
ProductResource.java
RestConfiguration.java
SupplierResource.java
fish.payara.service
AbstractService.java
CustomerService.java
DiscountService.java
LineItemService.java
OrderService.java
ProductService.java
SupplierService.java
fish.payara.service.producer
Other Sources
Dependencies
Non-classpath Dependencies
Java Dependencies
Project Files

IDE Integration

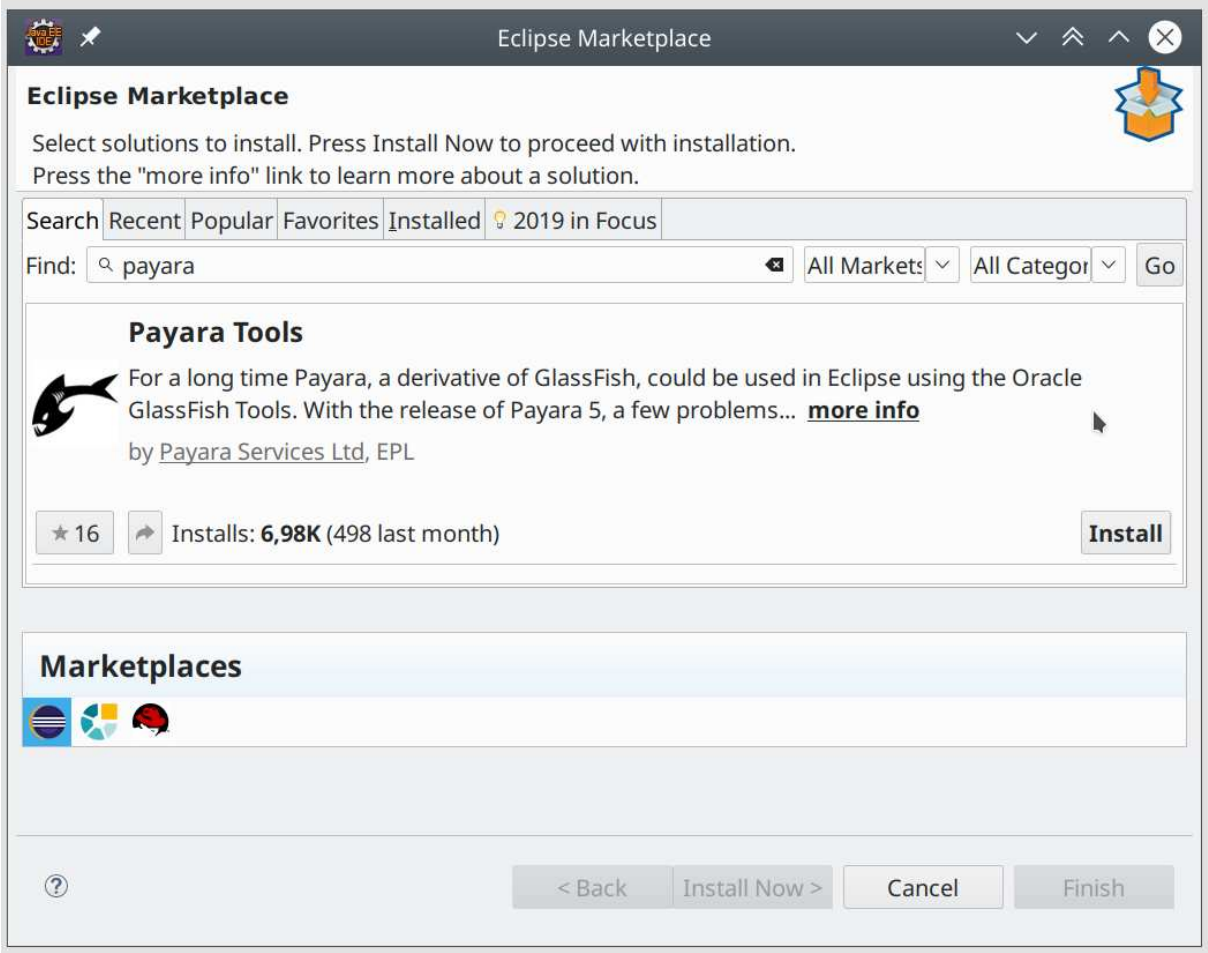
Eclipse Plugin

Payara Eclipse IDE Plugin

The Payara Tools plugin enables integration between Payara Server and Payara Micro into the Eclipse IDE. The Payara Tools plugin is available from the Eclipse Marketplace.

Installing Payara Tools Eclipse Plugin

1. Go to **Help** ▯ **Eclipse Marketplace…**
2. On the Marketplace window, type "**Payara**" and click the **Go** button on the right side.
3. Click the **Install** button.



4. Read and accept the license agreement.
5. You will see a **Security Warning**, click **Install Anyway**.

6. Finally, you will need to restart Eclipse.

Azul Payara Tools

- "Payara Server" - The Payara Eclipse plugin enables developers to easily configure Payara Server and manage applications from the Eclipse IDE.
- "Payara Micro" - Payara Eclipse plugin provides the support to create, run and debug Payara Micro applications from the Eclipse IDE. Payara Micro is the microservices-ready version of Payara Server.

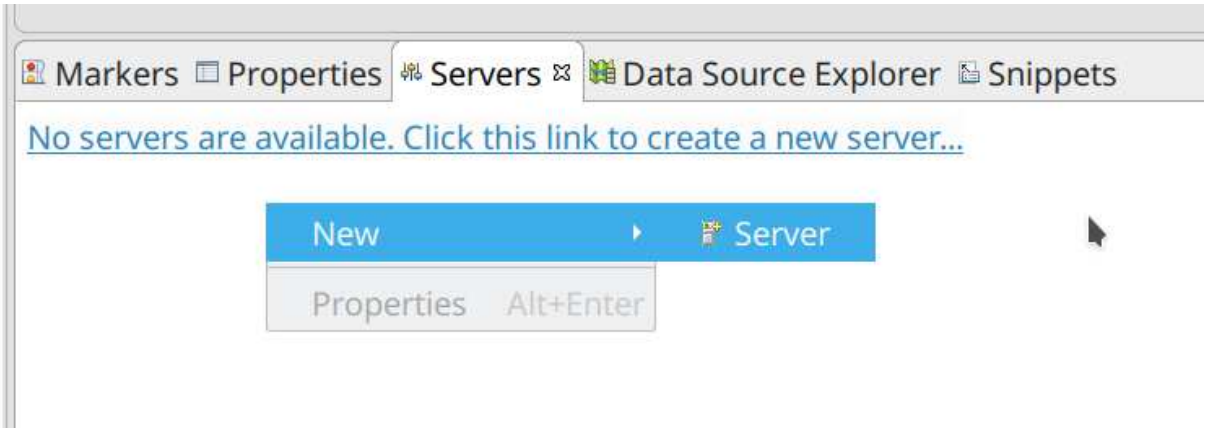
Payara Server Tools in Eclipse IDE

Using the Payara Tools Eclipse plugin, you can integrate Payara Server within the Eclipse IDE and perform the following tasks:

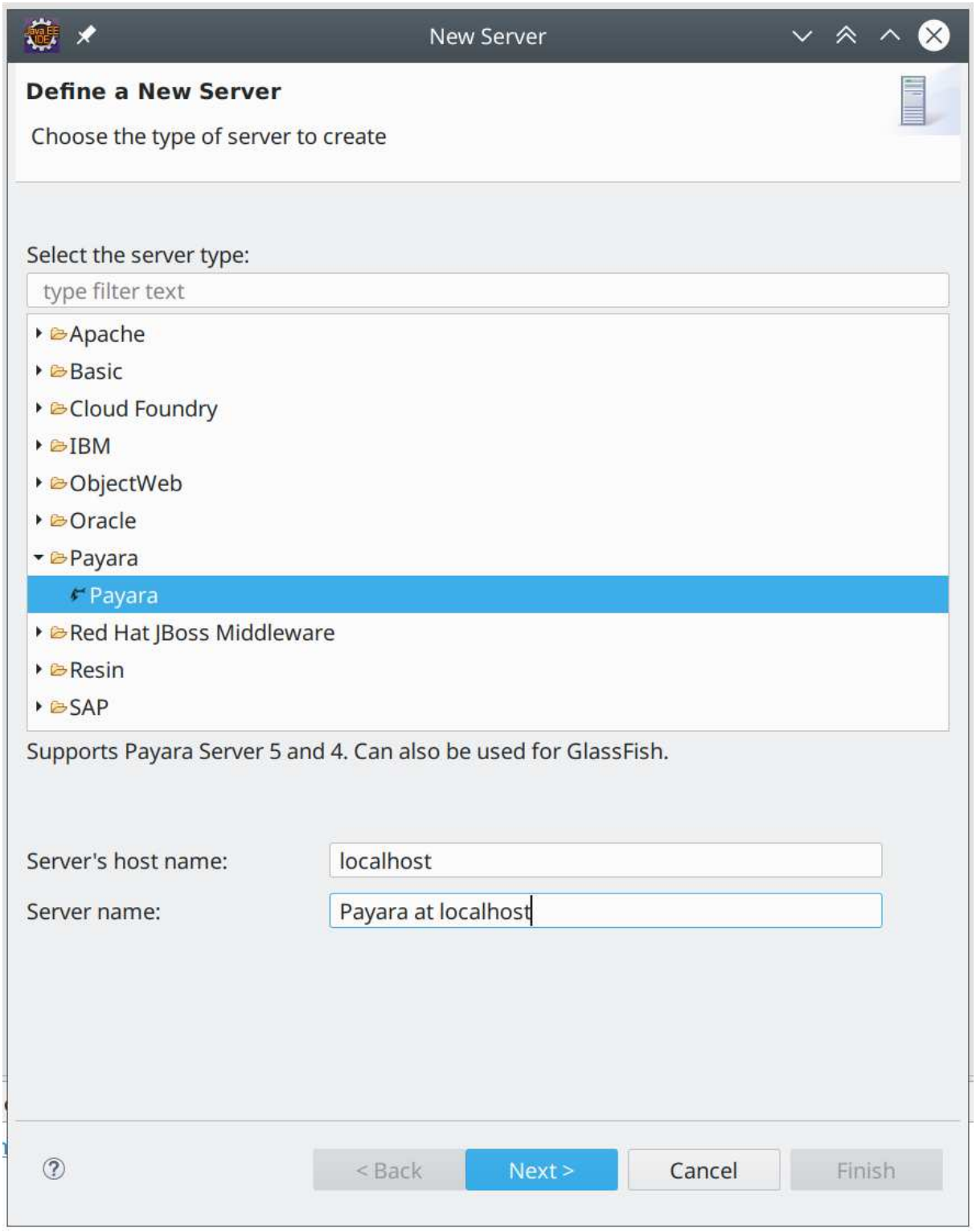
- Start and stop the Payara Server from within the IDE
- Define the Java runtime to be used by Payara Server
- Deploy an application from within the IDE
- Launch the admin console within the IDE
- View the server logs from the IDE
- Auto Publish & Hot Deploy your application

Adding Payara Server to Eclipse IDE

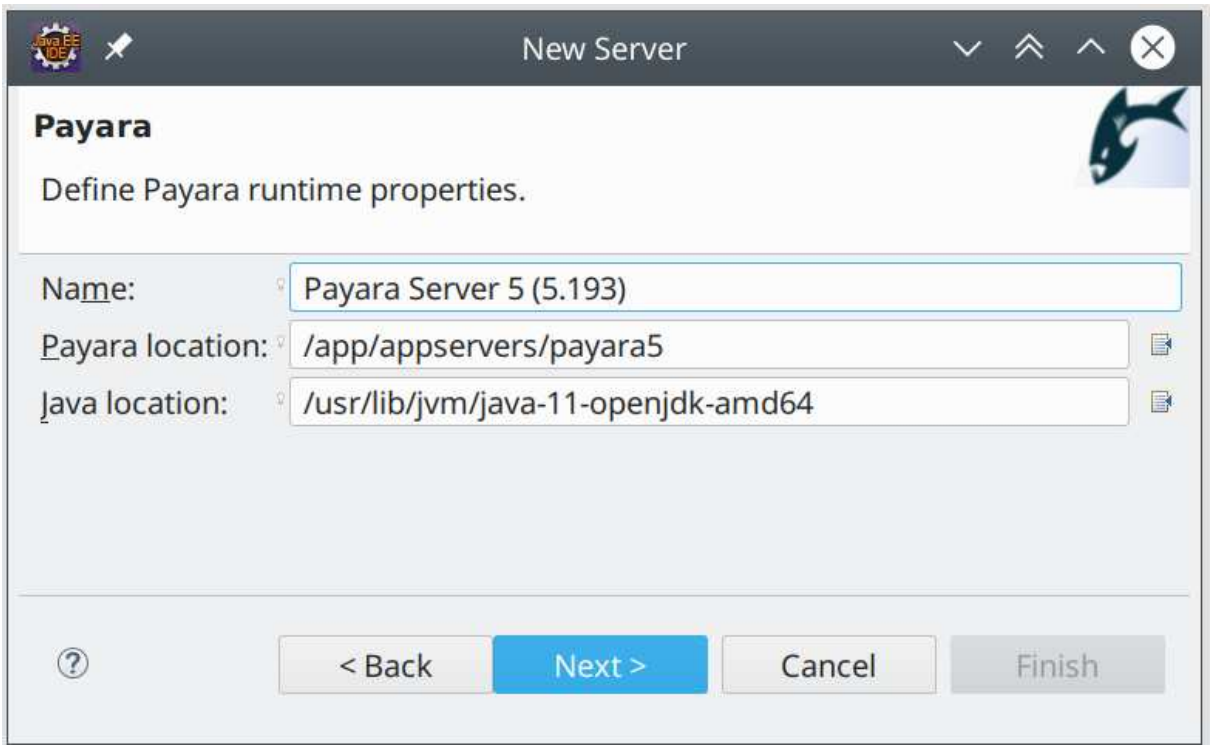
1. Open the **Servers** view (if not visible, it should be under menu **Window** ▯ **Show View** ▯ **Servers**).



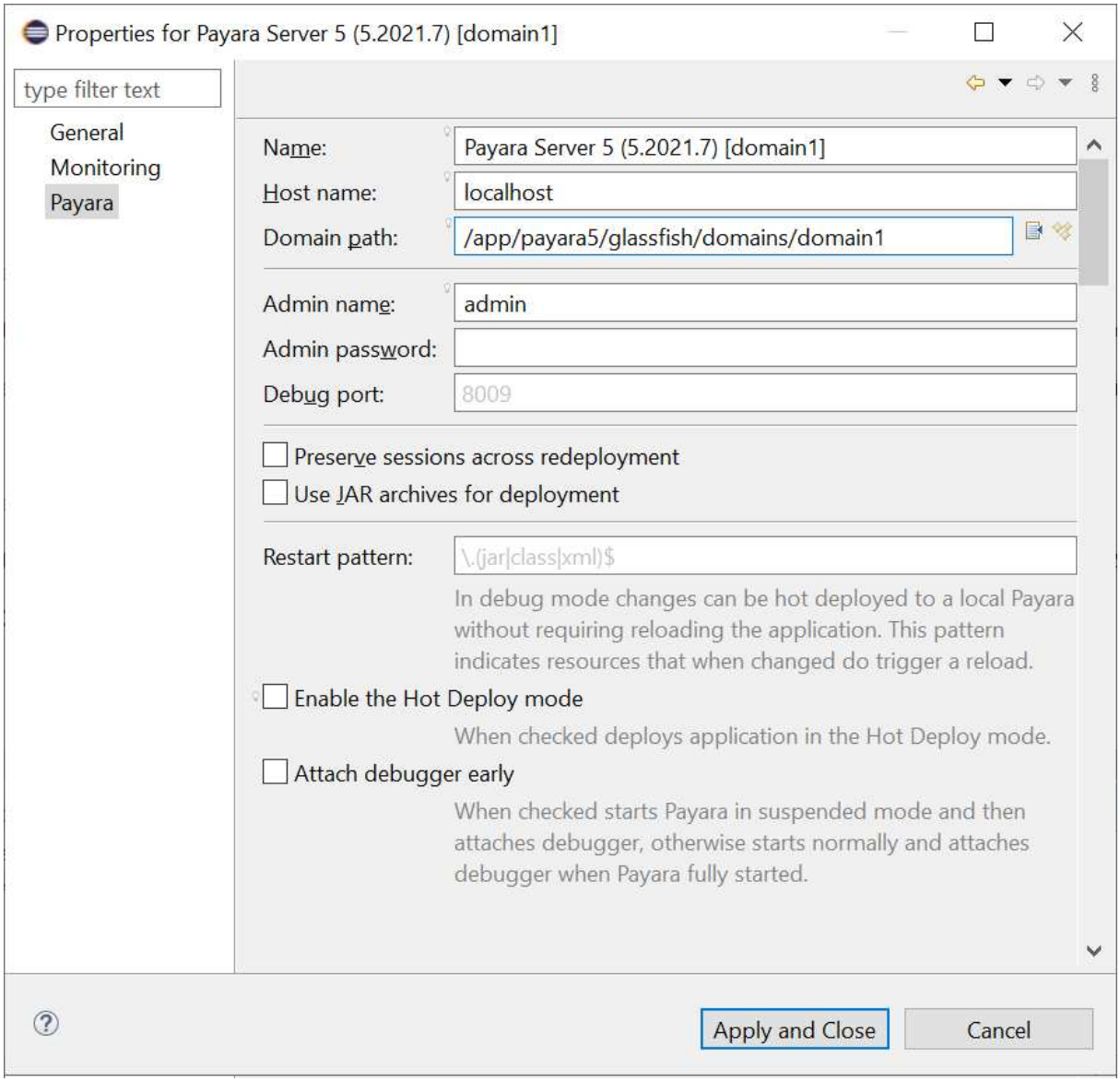
2. Select **Payara** ▯ **Payara** and set the **Host name** and some comprehensible **Server name**.



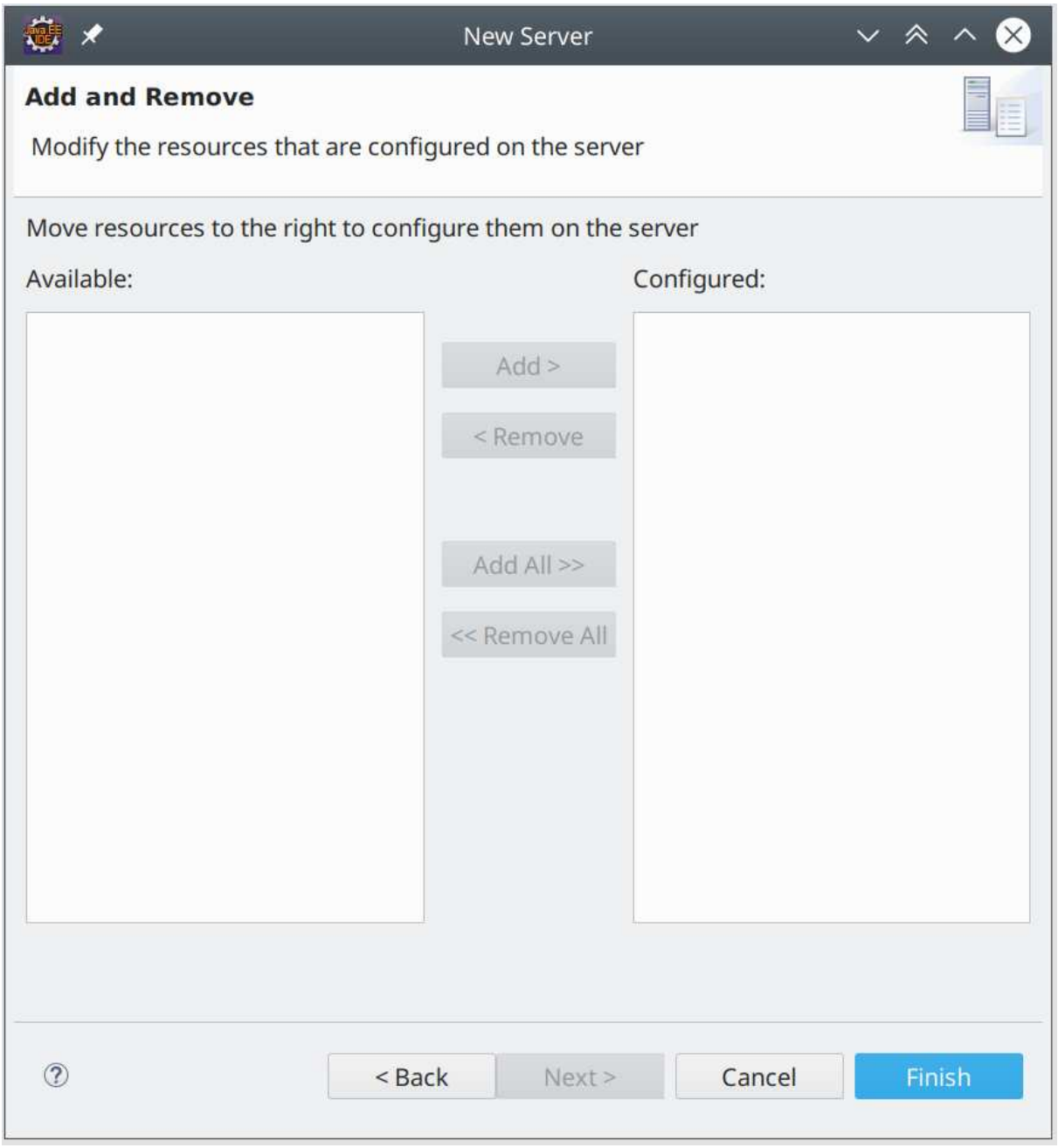
3. Then click **Next** to another window with server runtime configuration.



4. Then click **Next** again to window with the configuration of the domain. In this example we use domain without configured admin user and the server is not started in debug mode.

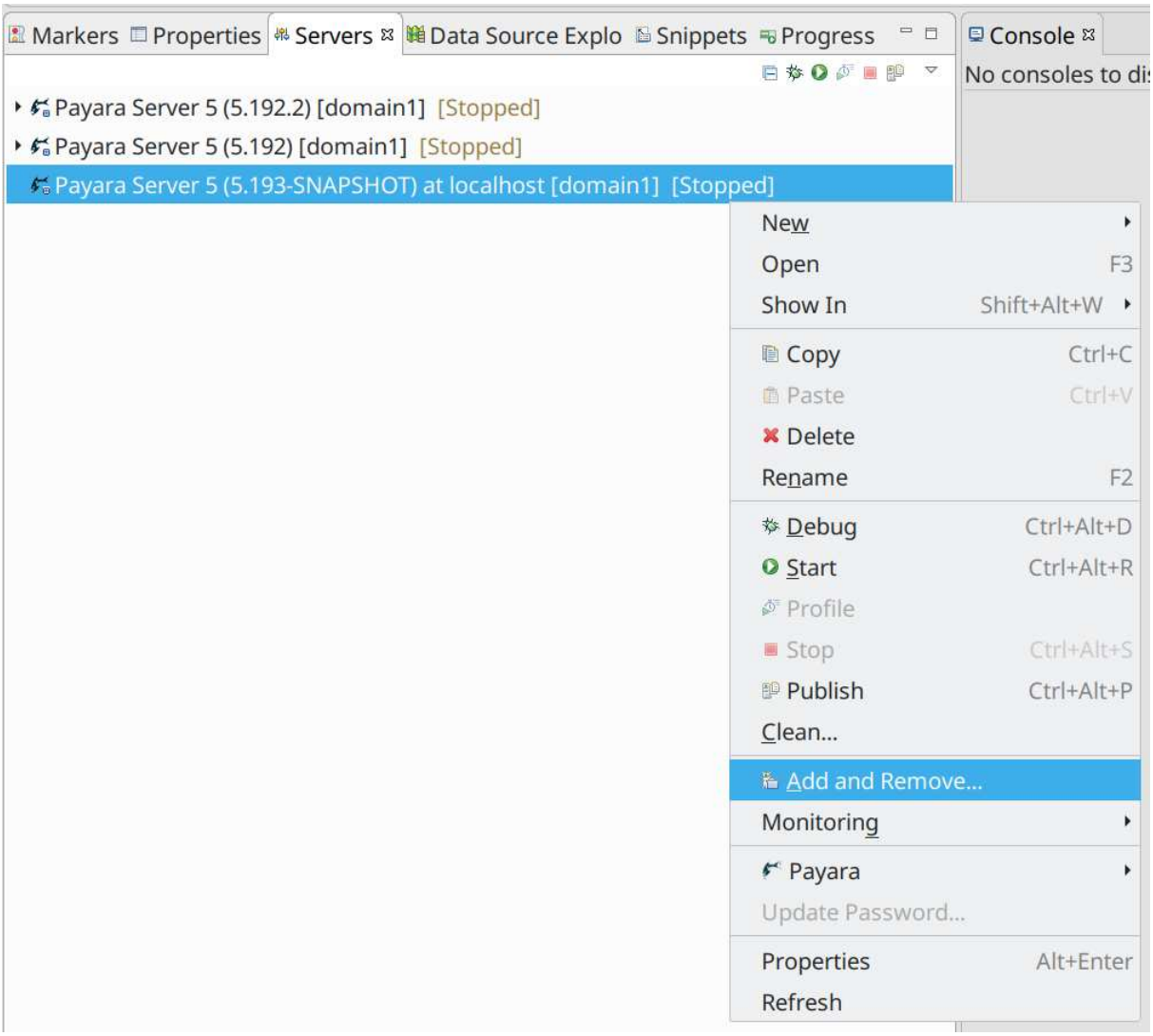


5. In the last step you can select resources which will be deployed on the server. Here we don't have any deployable projects in the workspace yet, so both panels are empty.

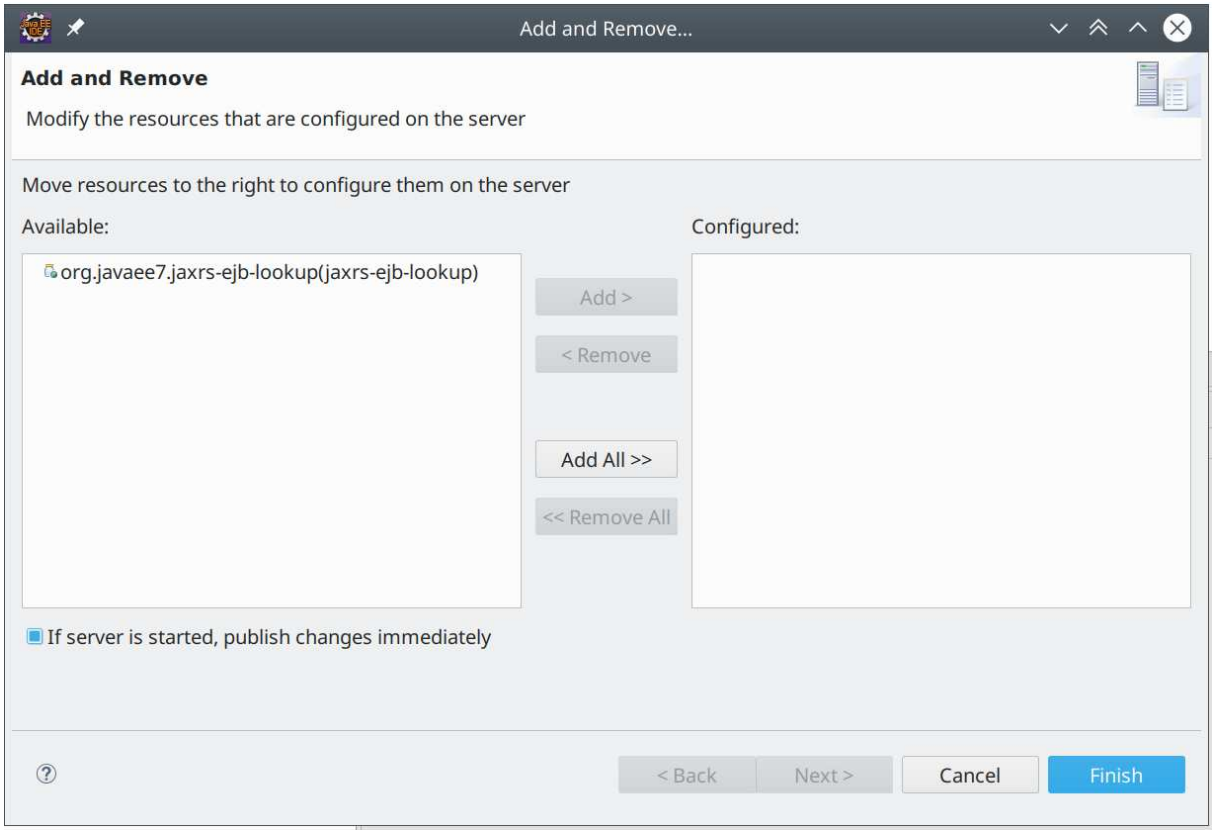


Adding Resources to Server

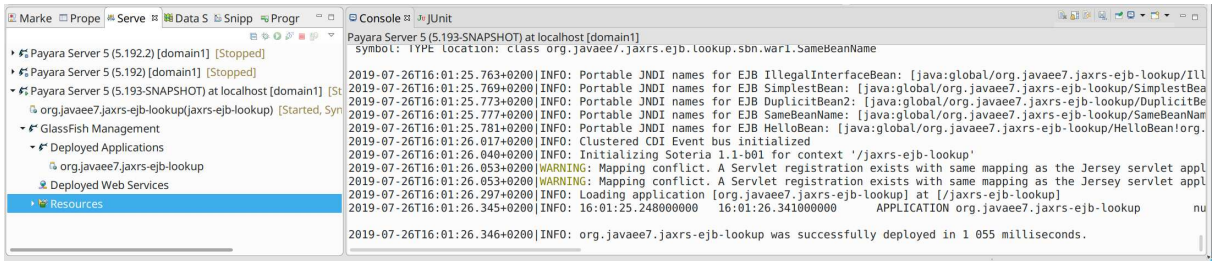
1. First import or create some project, deployable to the **Payara Server**. Here we use one of the Java EE 7 Samples .
2. Right-click the target server in the **Servers** view and click **Add and Remove…**



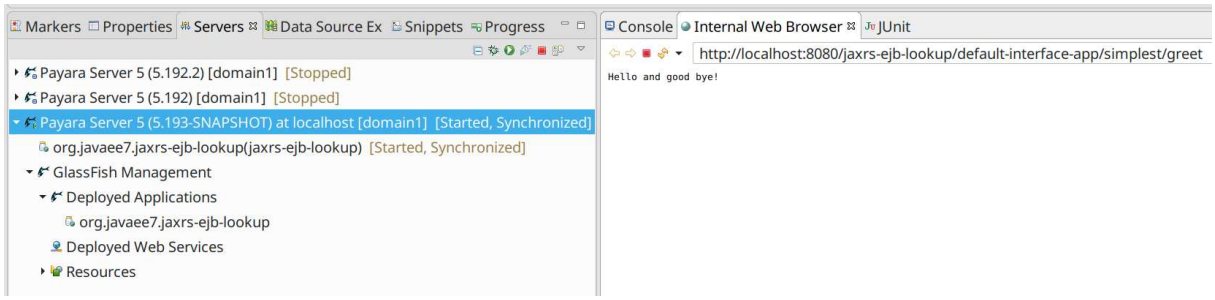
3. Select resources you want to be managed by the **Payara Tools**, add them to the right panel and click **Finish**.



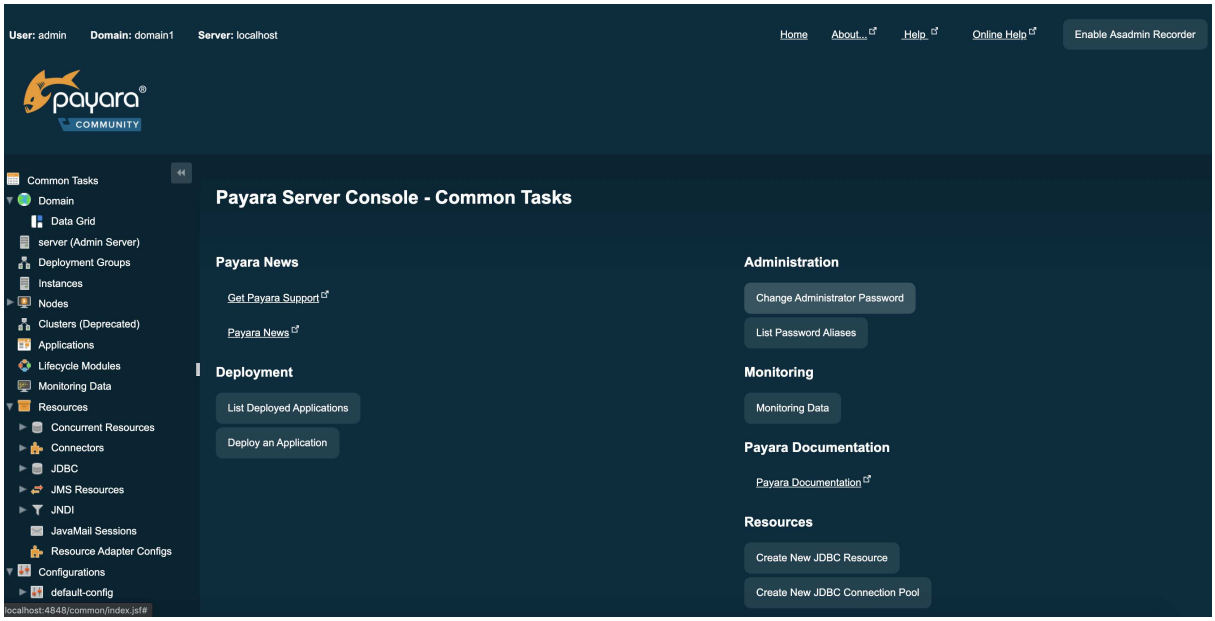
4. Now you can watch the deployment of the application in the **Console** view.

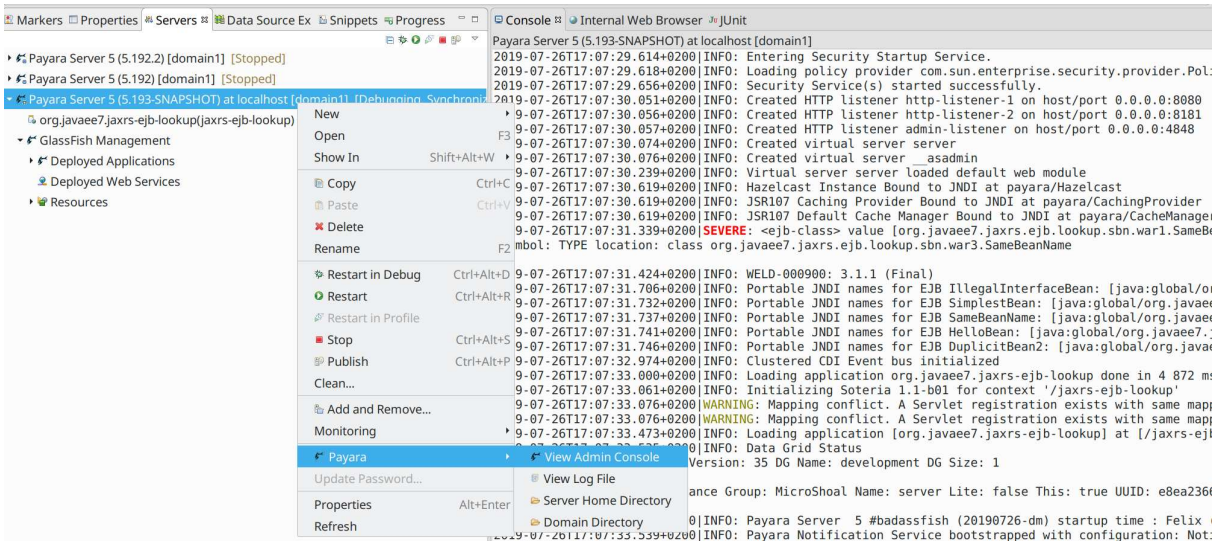


5. If the deployment was successful, you can use the application. Here we simply opened the internal Eclipse's browser and sent the HTTP GET request - and the application responded.



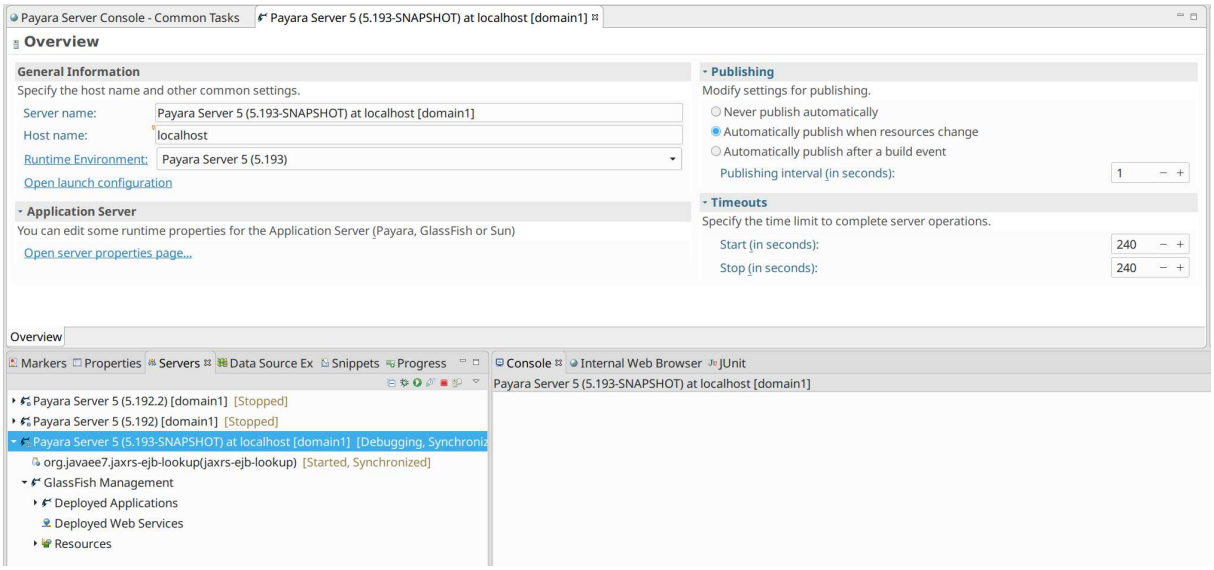
6. You can also access the administration console directly from the Eclipse Browser. Just right-click the target server in the **Servers** view and click **Payara View Admin Console**





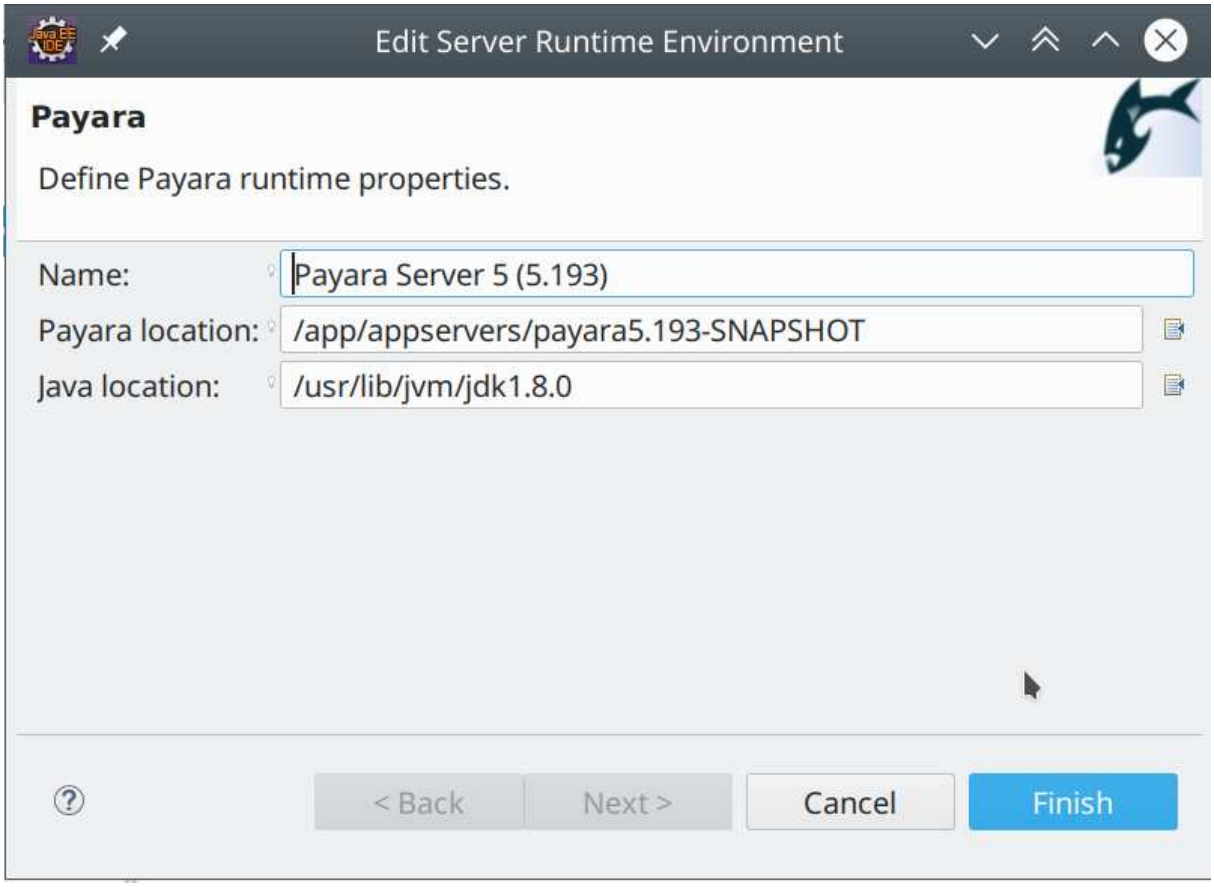
Runtime Reconfiguration

1. In the **Servers** view double-click the server you want to reconfigure. The **Server Overview** will open.



2. Click on **Runtime Environment** and you will see well known window.

WARNING	Changing the runtime may result in unusable state of the server because of incompatible configuration.



Deploy Applications in a Local Docker Container

The Payara Server tools for Apache NetBeans include Docker integration, making it seamless to deploy exploded applications to a Payara Server instance running within a local Docker container.

To take advantage of this feature, follow these steps below:

1. Register a new Payara Server instance by performing the following:
    a. Open the **Servers** view (if not visible, it should be under menu **Window** → **Show View** → **Servers**).
    b. Select **Payara** → **Payara** and set the **Host name** and some comprehensible **Server name**.
    c. Then click **Next** to another window with server runtime configuration and browse and locate the Payara Server installation directory with the same version as the one running in the Docker container.
    d. Then click **Next** again to window with the configuration of the domain. Enter the domain name, host, and necessary credentials.
    e. For Docker integration, choose the "Docker Volume" option and provide the `HOST_PATH` (e.g., `D:/test`) and `CONTAINER_PATH` (e.g., `/mnt/test`) mapping details.

    This mapping should align with the exploded application path or parent path of exploded application.

    f. In the last step you can select resources which will be deployed on the server. You may add them later.
2. Run the Payara Server Docker container using the following command:

```
docker run -p 8080:8080 -p 4848:4848 -p 9009:9009 -v HOST_PATH:CONTAINER_PATH
payara/server-full:{release-id}
```

Replace `HOST_PATH` with the actual host path and `CONTAINER_PATH` with the

desired container path.

The `HOST_PATH` points to the location on your host machine, and `CONTAINER_PATH` is the corresponding location inside the container.

3. Create your web application and deploy it to the registered Payara Server instance.

Deploy Applications in a WSL Environment

Another noteworthy feature is the integration of the Payara Server Tools with Windows Subsystem for Linux (WSL).

This integration empowers developers to manage and deploy exploded applications to a Payara Server instance running within WSL.

Here's how you can make use of this feature:

1. Start the Payara Server instance within WSL:
   a. Run the following command to start the server:

```
./asadmin start-domain
```

   b. Set a password for the admin user:

```
./asadmin change-admin-password
```

   c. Enable secure admin:

```
./asadmin enable-secure-admin
```

   d. Restart the Payara Server domain:

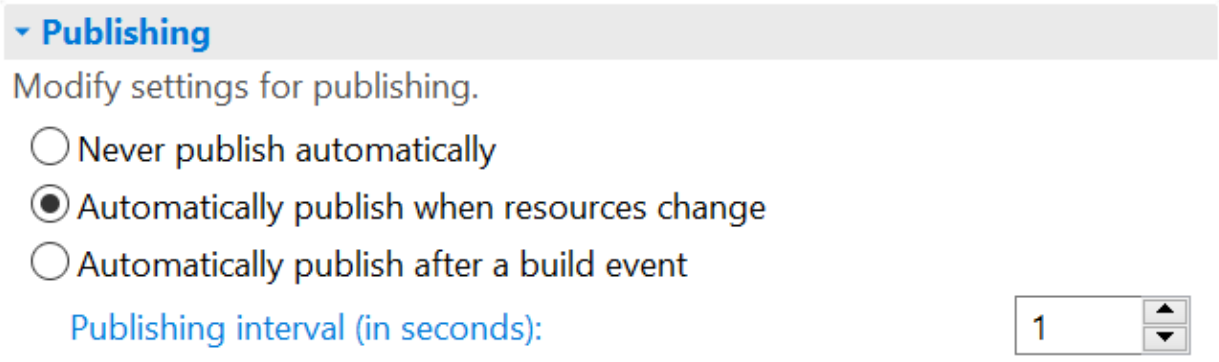
```
./asadmin restart-domain
```

   e. Fetch the IP address of WSL environment using the command `hostname -I` and enter it in the `host name` field in the Eclipse IDE settings.

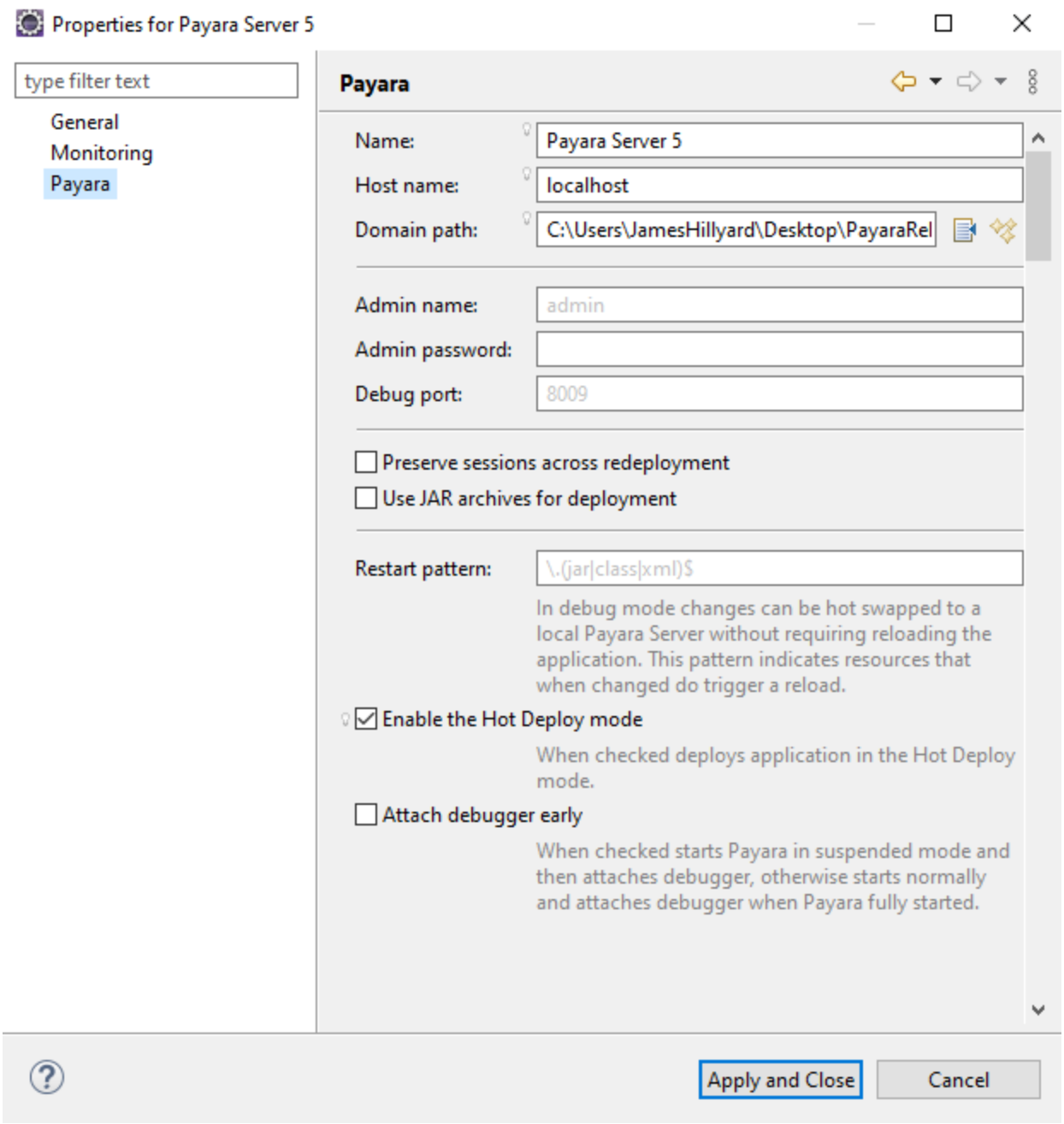
2. Register the WSL-based Payara Server instance in VS Code:
    a. Open the **Servers** view (if not visible, it should be under menu **Window** → **Show View** → **Servers**).
    b. Select **Payara** → **Payara** and set the **Host name** and some comprehensible **Server name**.
    c. Then click **Next** to another window with server runtime configuration and browse and locate the Payara Server installation directory with the same version as the one running in the Docker container.
    d. Then click **Next** again to window with the configuration of the domain. Enter the domain name, host name (fetched via the command `hostname -I` in WSL instance), and necessary credentials.
    e. For WSL integration, choose the "WSL" instance type.
    f. In the last step you can select resources which will be deployed on the server. You may add this setting later.
3. Create your web application and deploy it to the registered Payara Server instance.

Hot Deploy and Auto Deploy Settings

To enable **Auto Deploy** from the Eclipse IDE, open The **Server** panel, double-click the server name and expand the **Publishing** section to reconfigure the setting.



Hot Deploy mode can be enabled from the Payara Server properties page to boost the deployment performance which reuses the existing application instance relative to the modified source:



See Also

"Hot Deploy and Auto Deploy"

Payara Micro Tools in Eclipse IDE

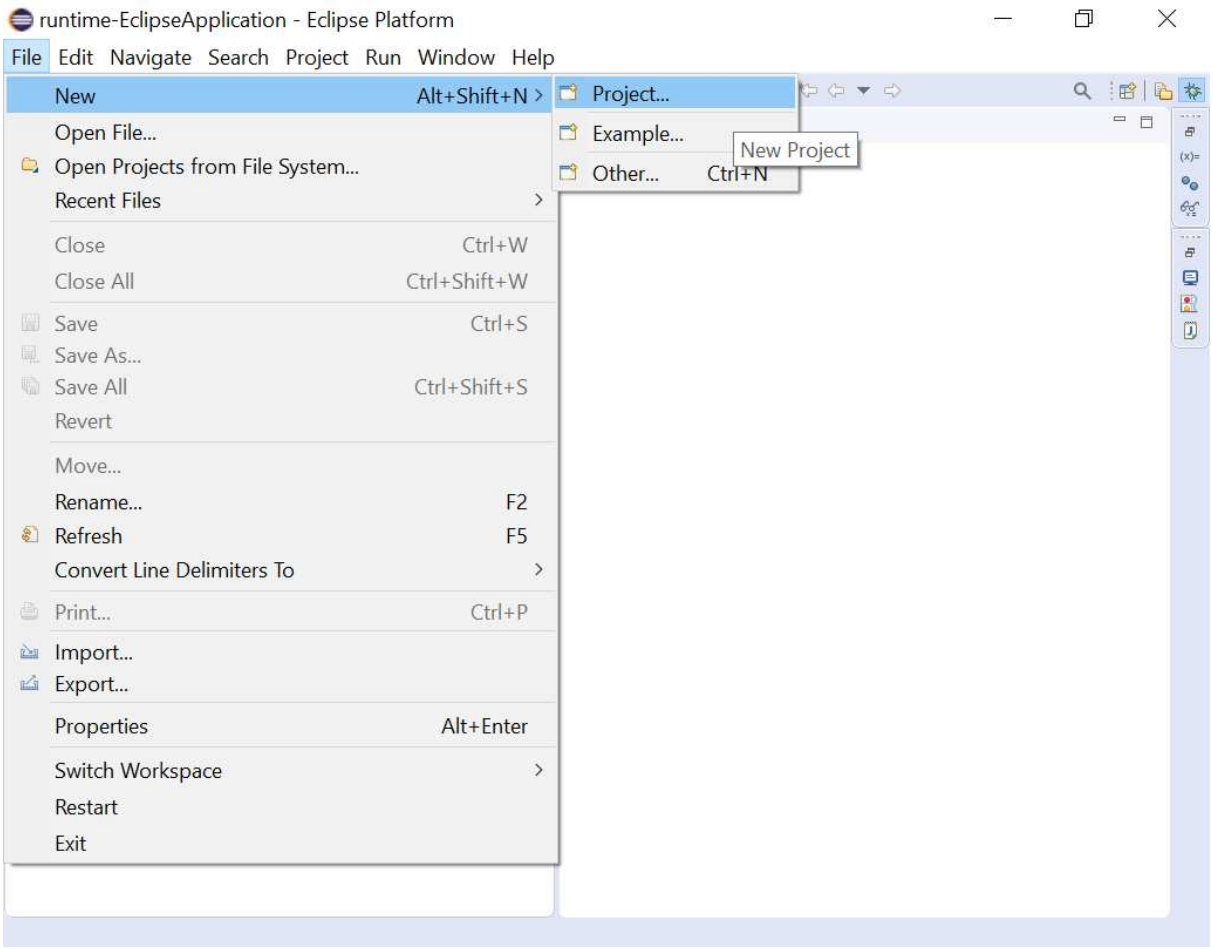
Using the Payara Eclipse Tools, you can integrate Payara Server within the Eclipse IDE and perform the following tasks:

- Run Payara Micro from within the IDE

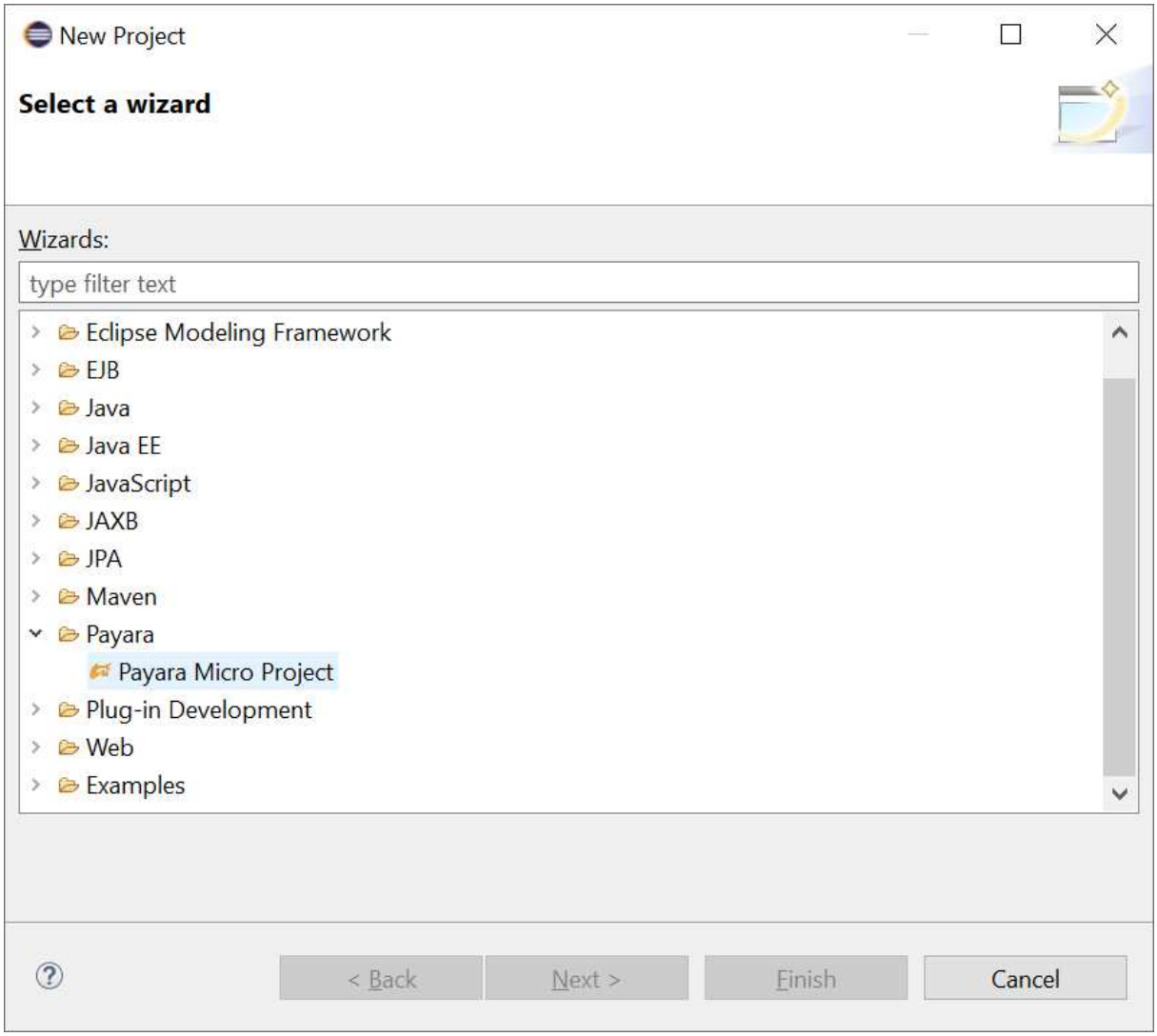
- Control Payara Micro properties from within the IDE
- Deploy applications to Payara Micro within the IDE
- Auto Publish & Hot Deploy your application

Creating A New Payara Micro Maven Application

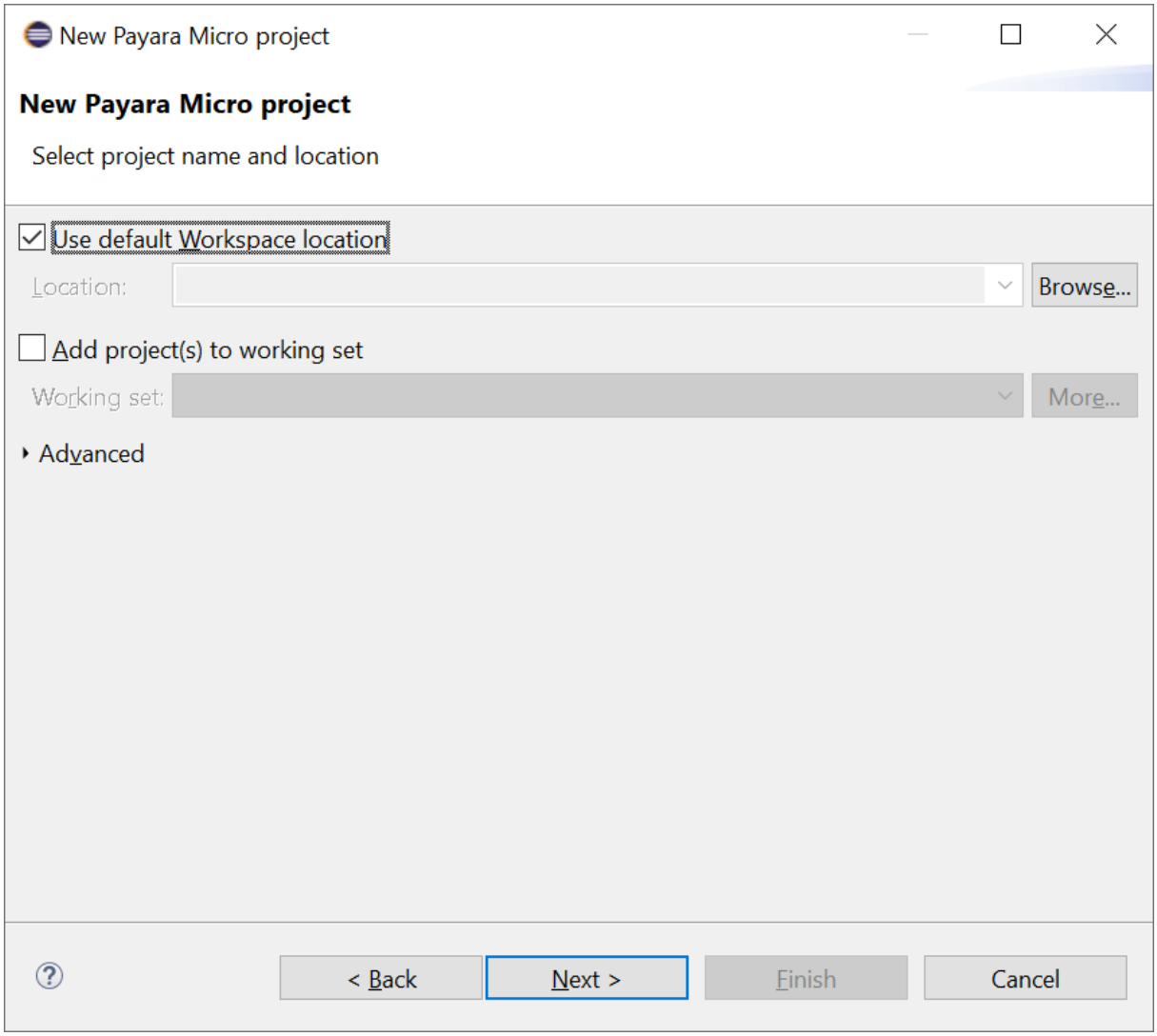
1. Simply start up Eclipse IDE and then Choose **File > New > Project** from the main menu or press Ctrl+N to get the list of available projects:



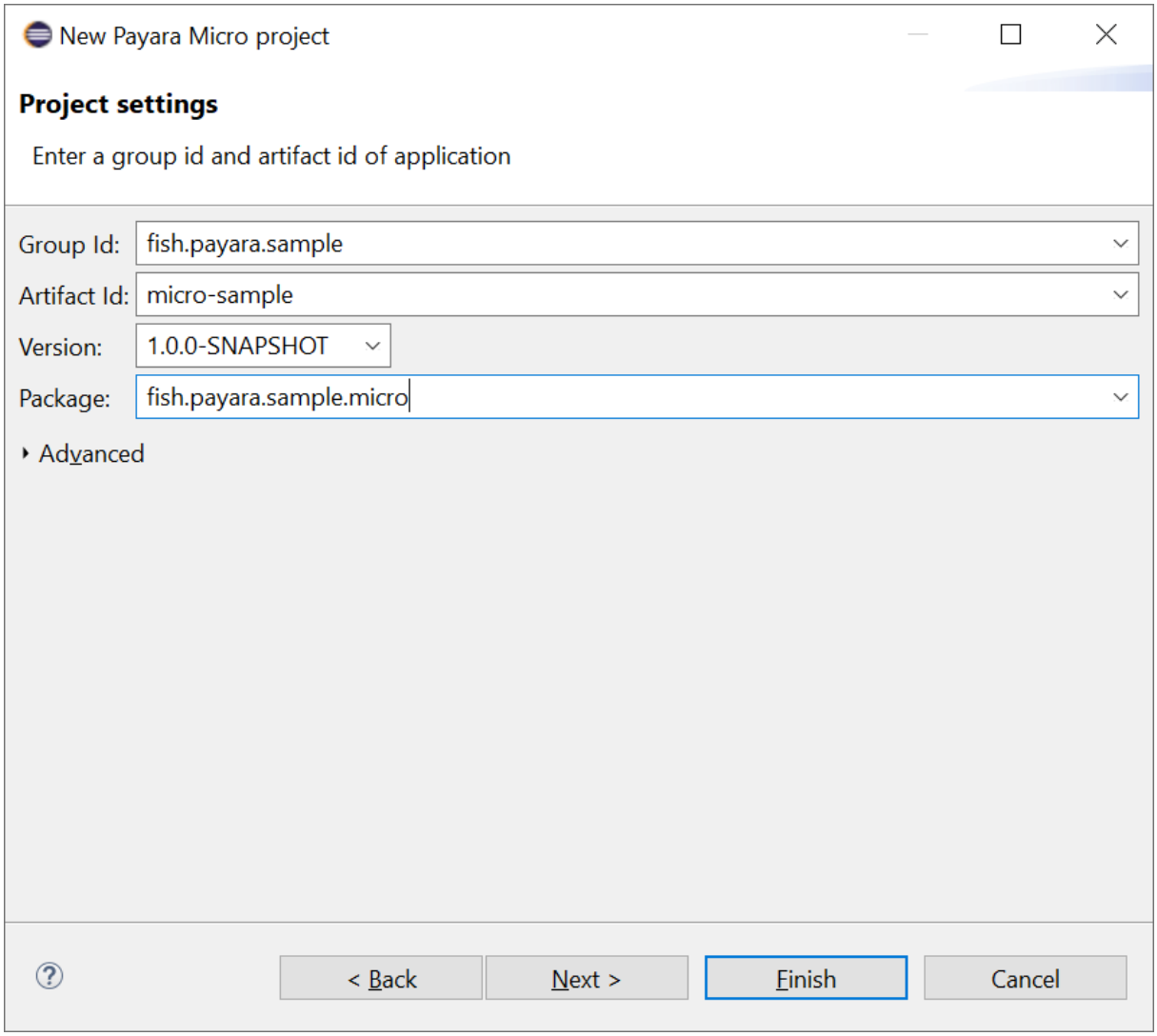
2. Now, select the Payara Micro Project type to create the application:



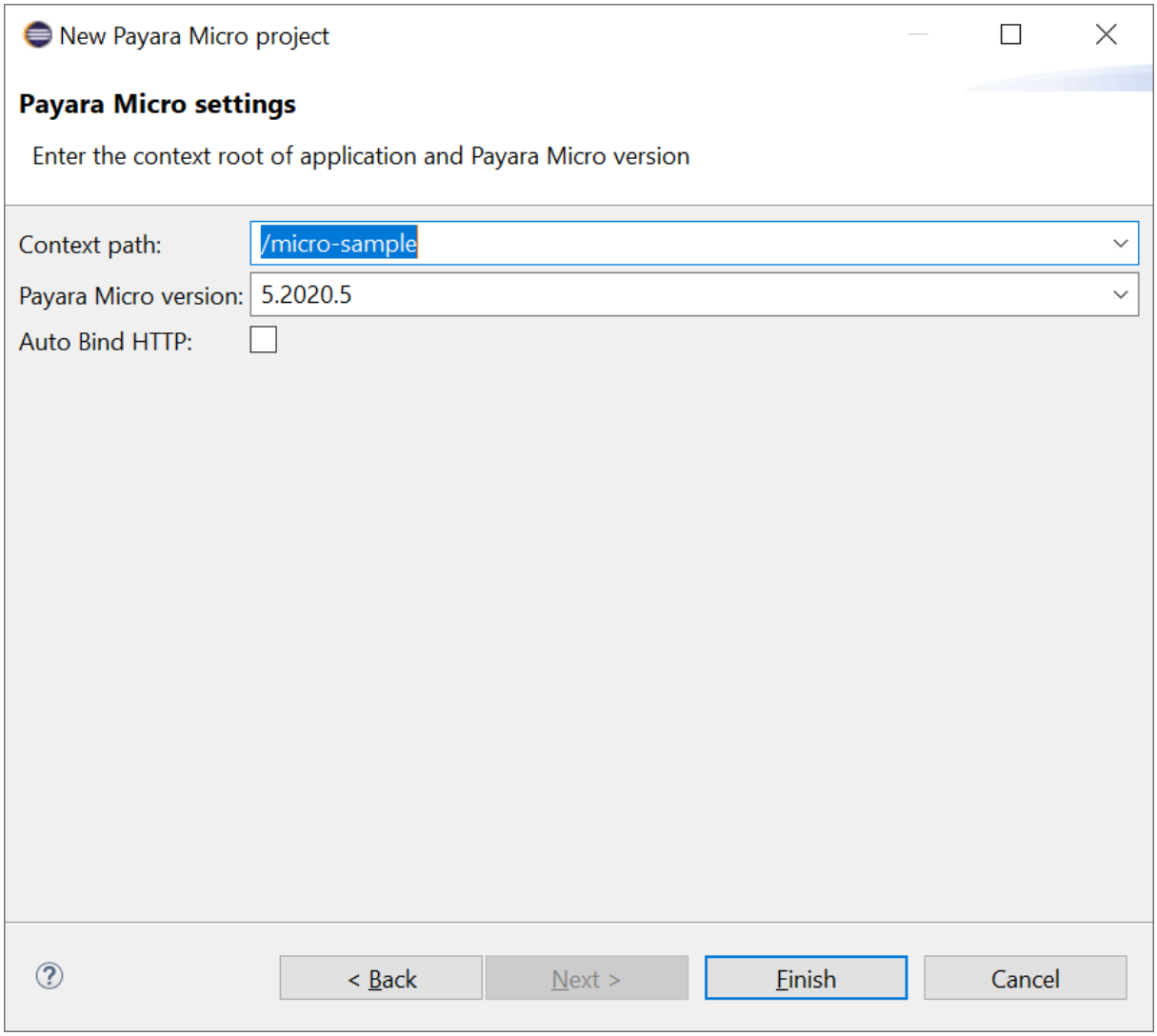
3. Select the destination folder, where the project will be generated. You may also use the default workspace location:



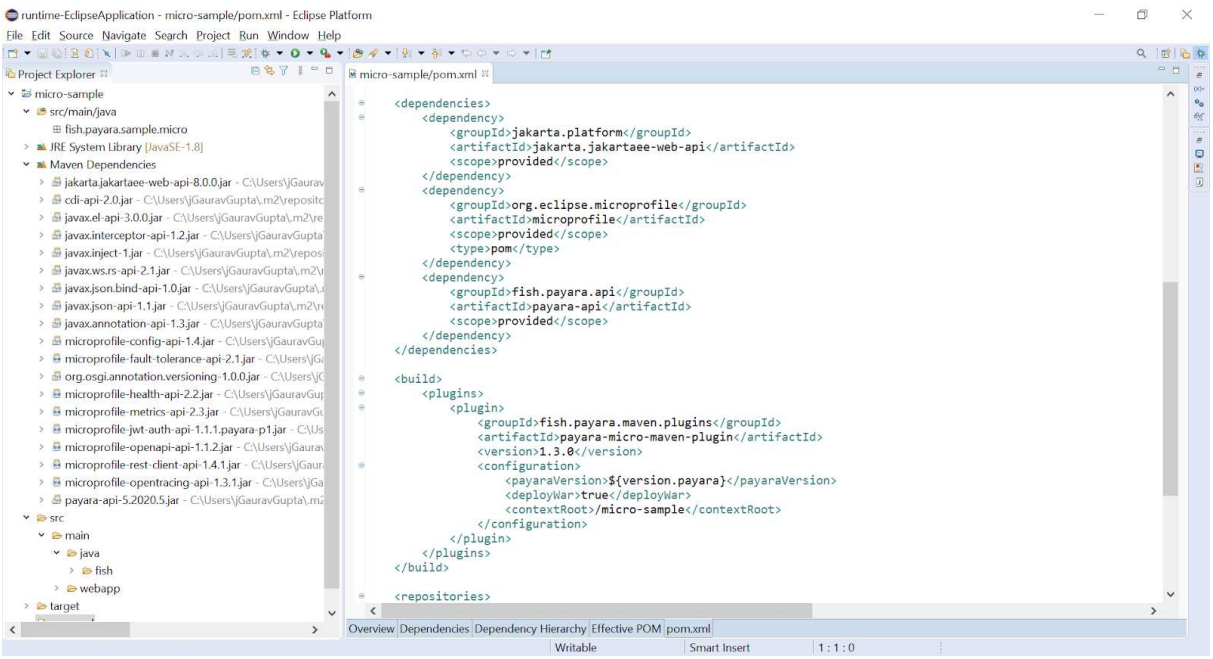
4. Enter the groupId, artifactId, etc to set up the project, and then click **Next**:



5. Enter the Payara Micro application settings e.g context path of the application and Payara Micro version and click the **Finish** button:



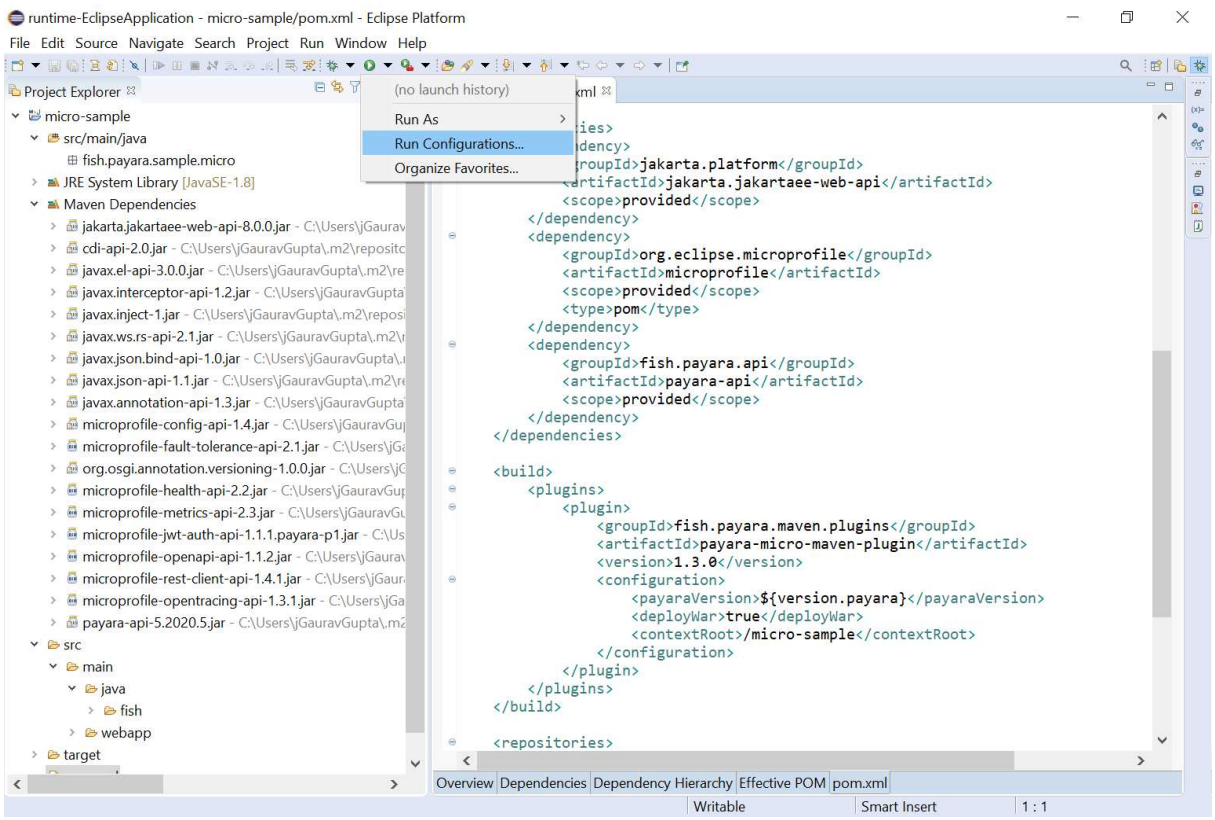
6. The IDE creates the Payara Micro application using the Payara Micro maven archetype. As you can see, the project has a basic structure with `src\main\java` and `src\main\webapp` folders:



Running The Payara Micro Application

Eclipse IDE needs a run configuration to build the artifacts and run/debug the Payara Micro application.

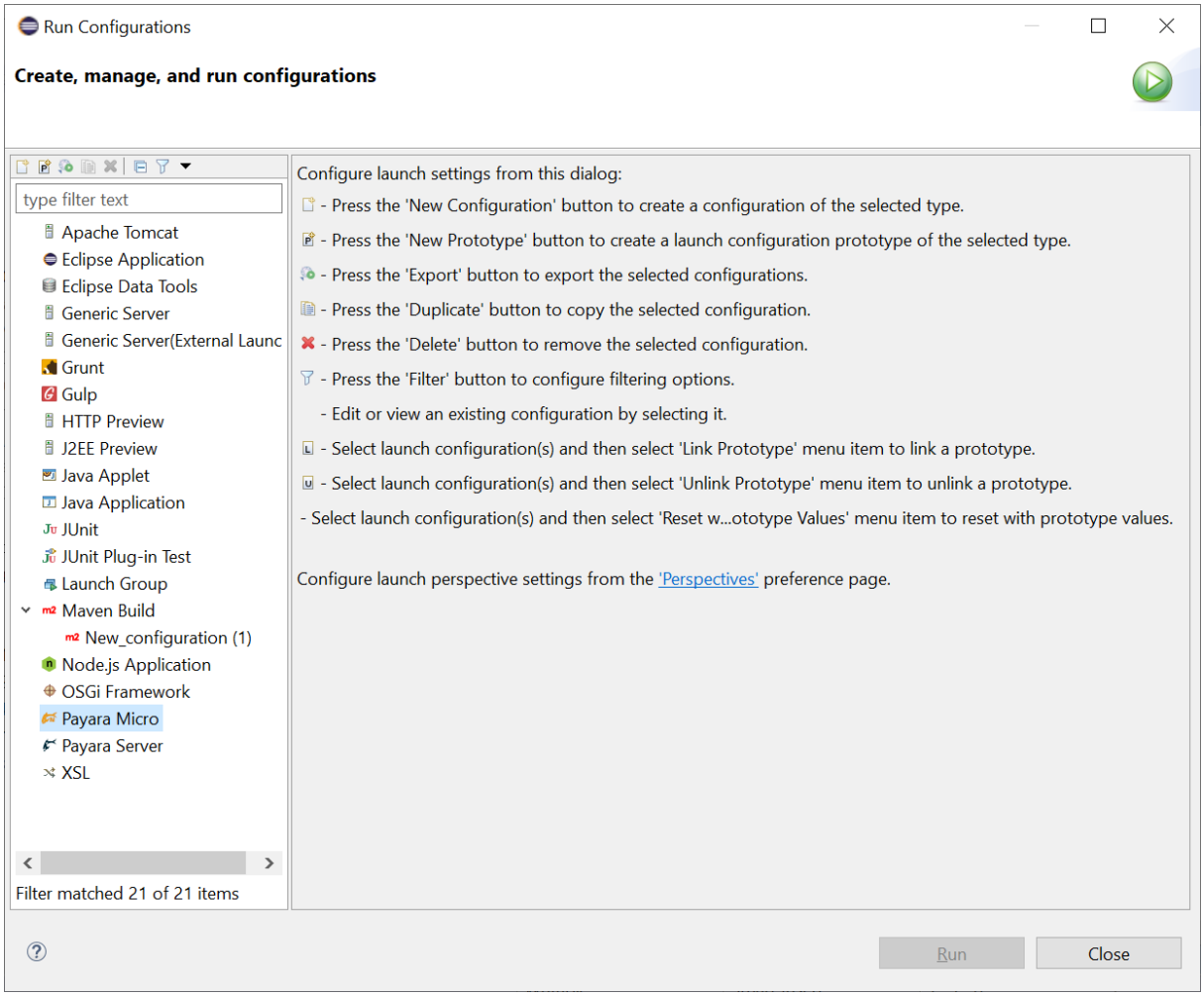
1. To create the Payara Micro Run configuration, select the **Run > Run Configurations…** from the main menu or toolbar:



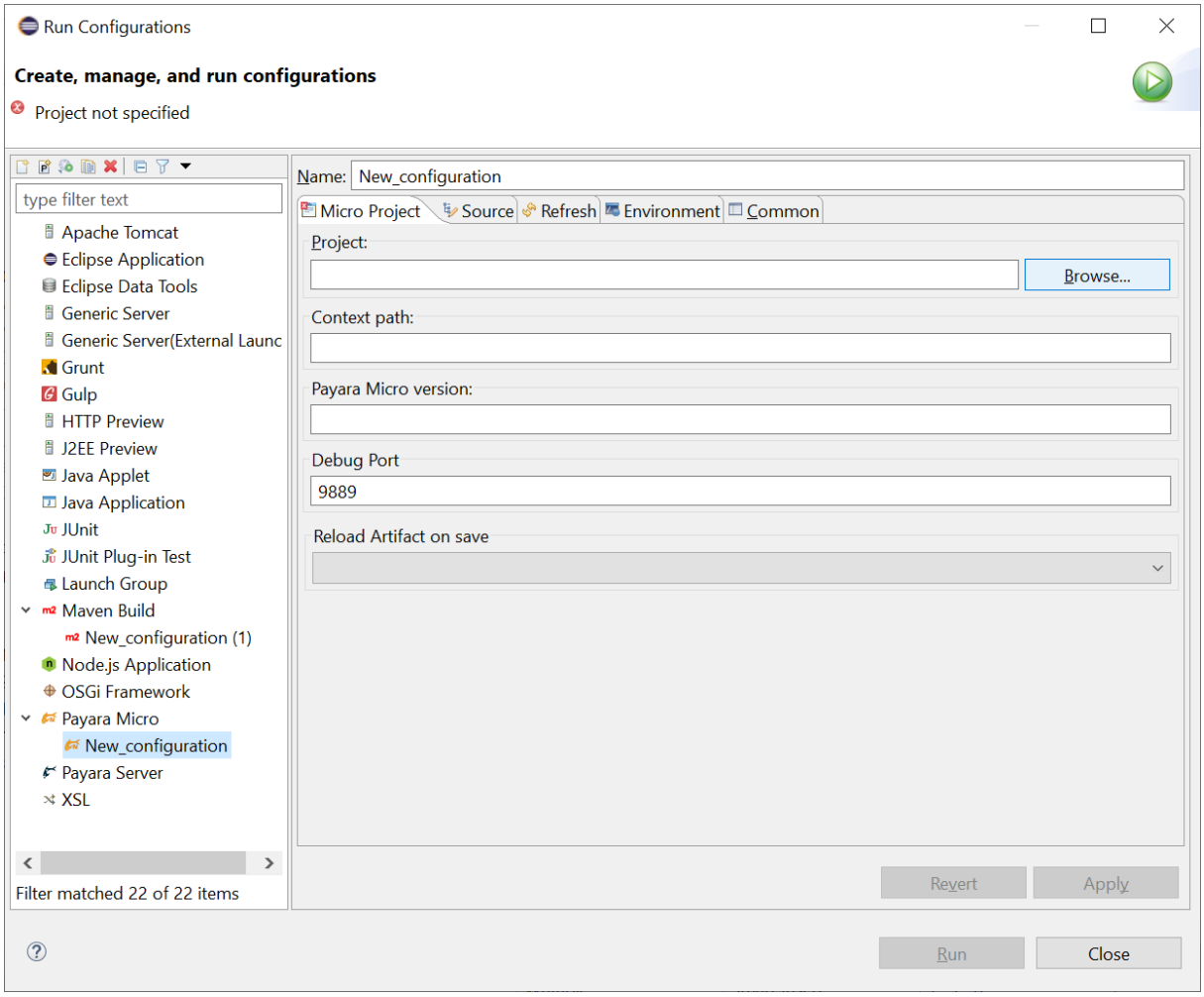
2. The Payara Micro Run configuration dialog allows you to specify Payara Micro settings and environment variables.

NOTE	Payara Micro setting specified by **payara-micro-maven-plugin** in the **pom.xml** takes precedence over the **Run Configuration**.

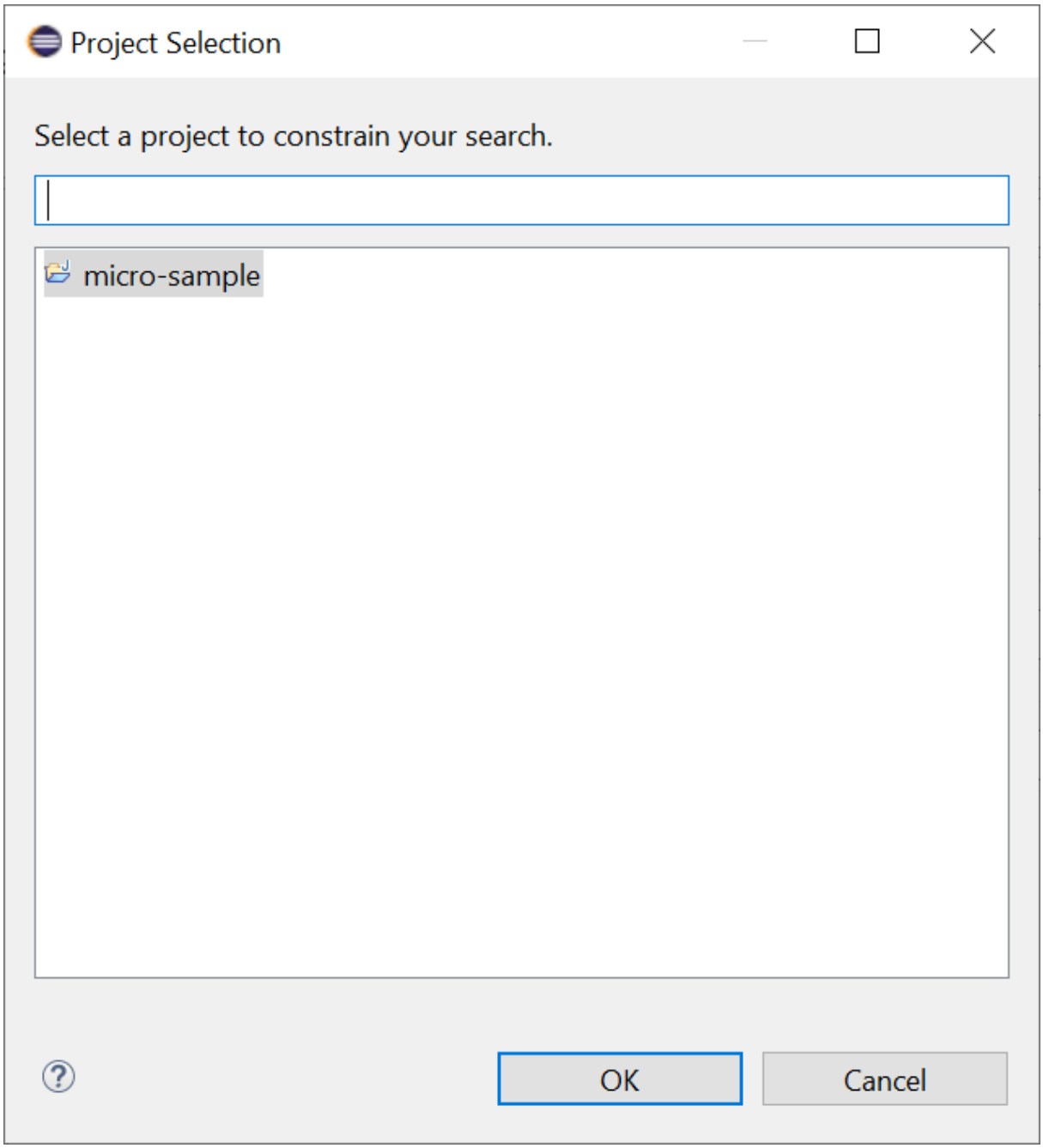
3. Select Payara Micro and double click it or click on the **New configuration** button to create a new configuration :



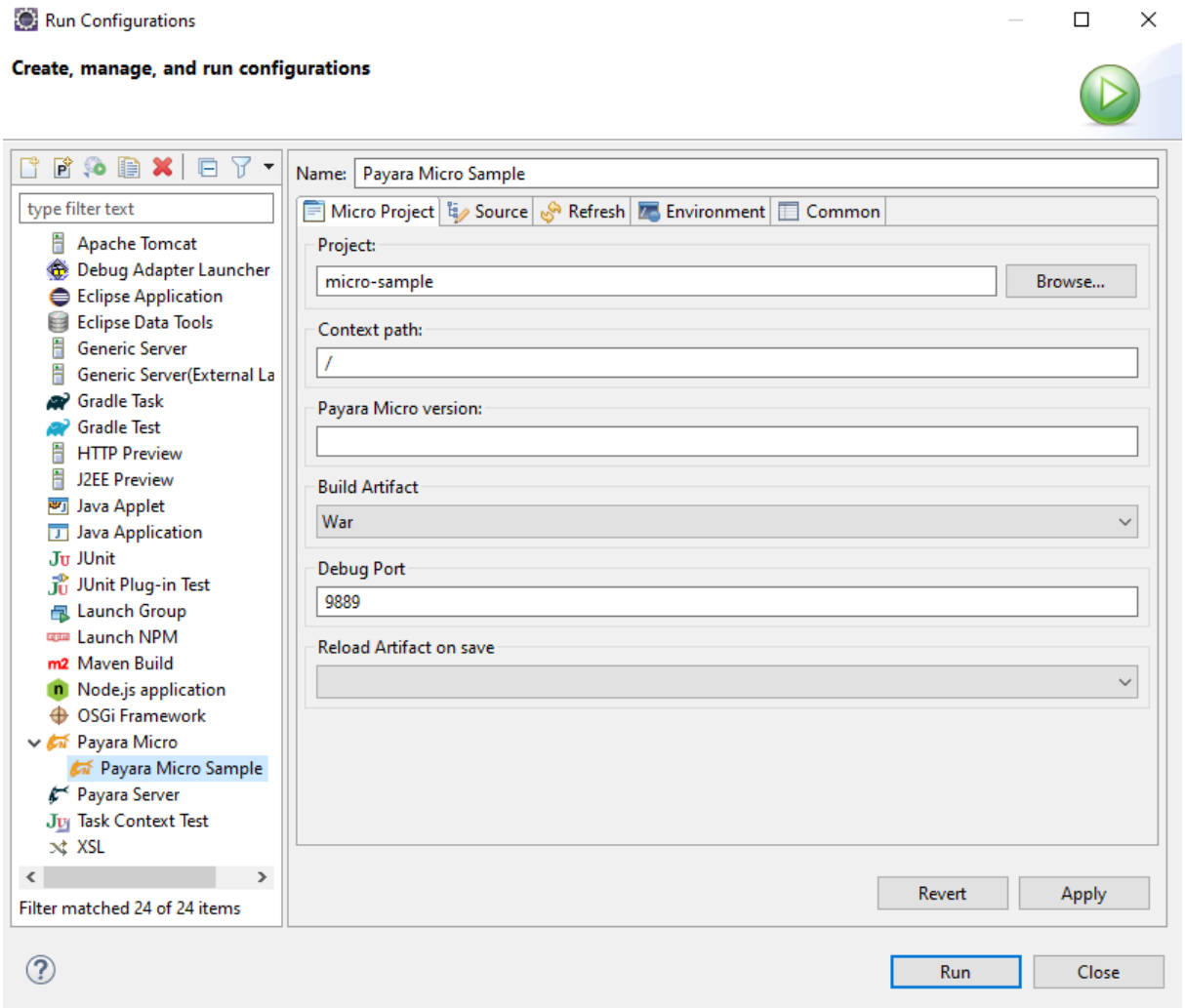
4. Associate the workspace project with the run configuration:



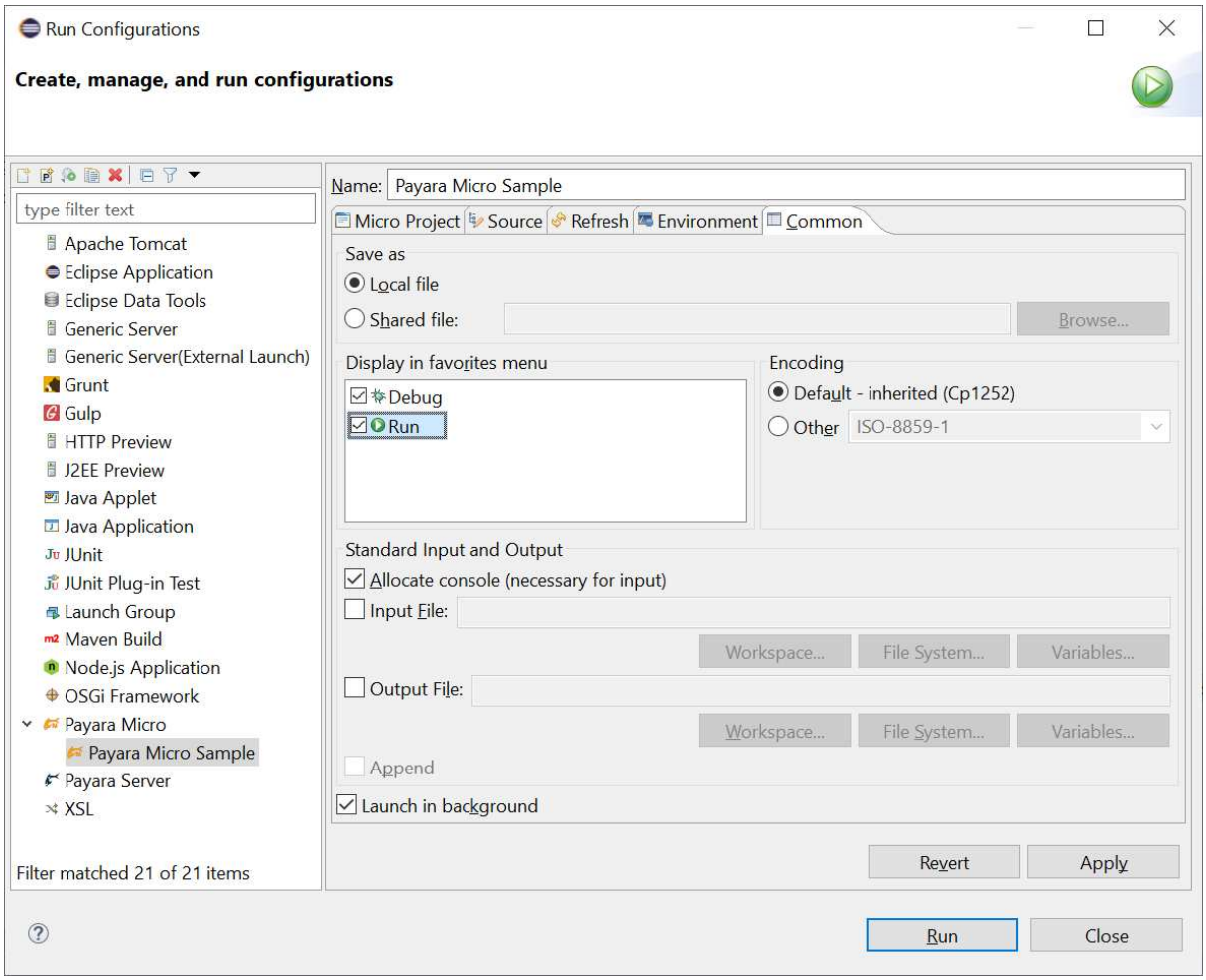
5. Click the browse button and select the project from the **Project Selection** dialog:



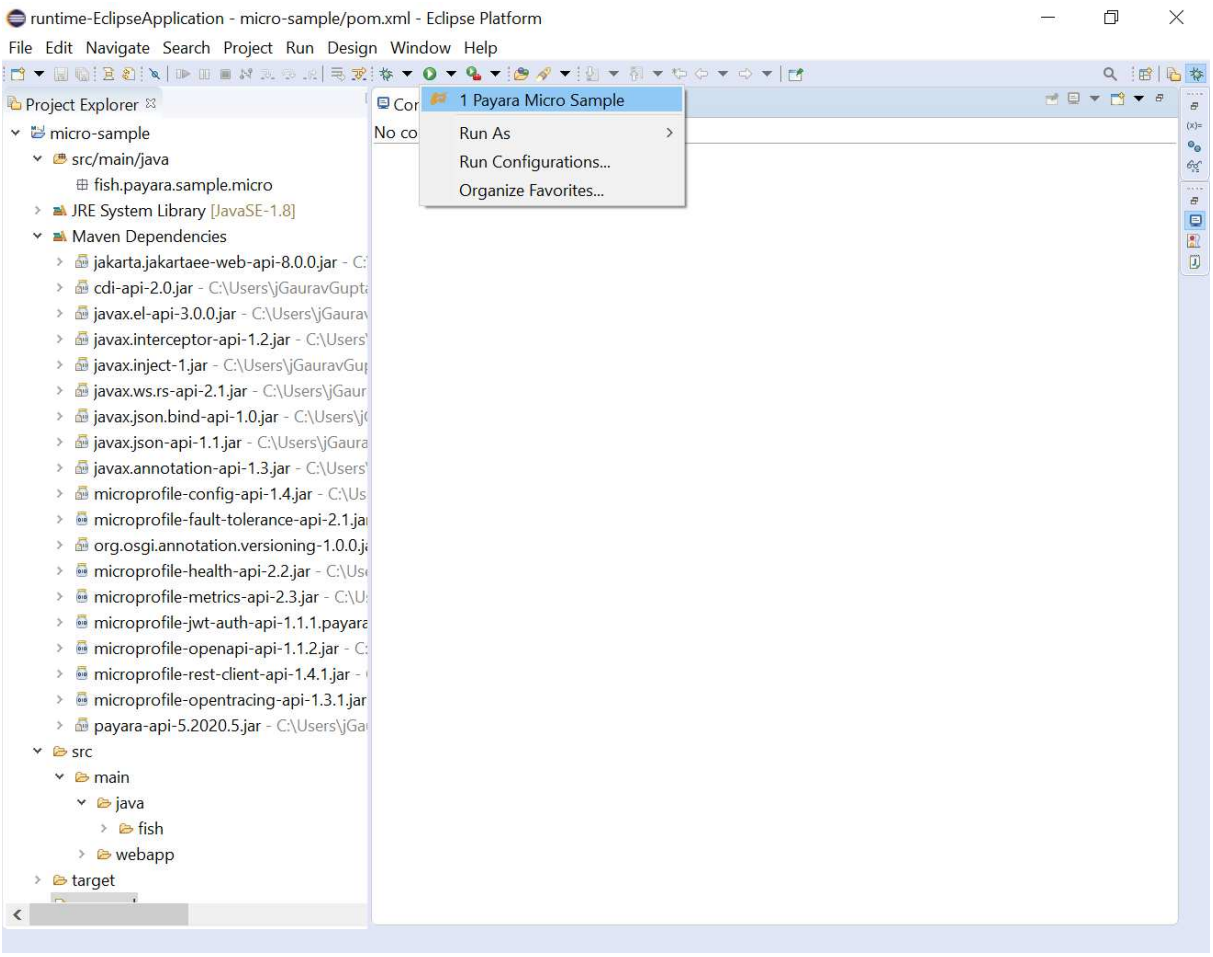
6. Enter the configuration name and choose the **Build Artifact** type



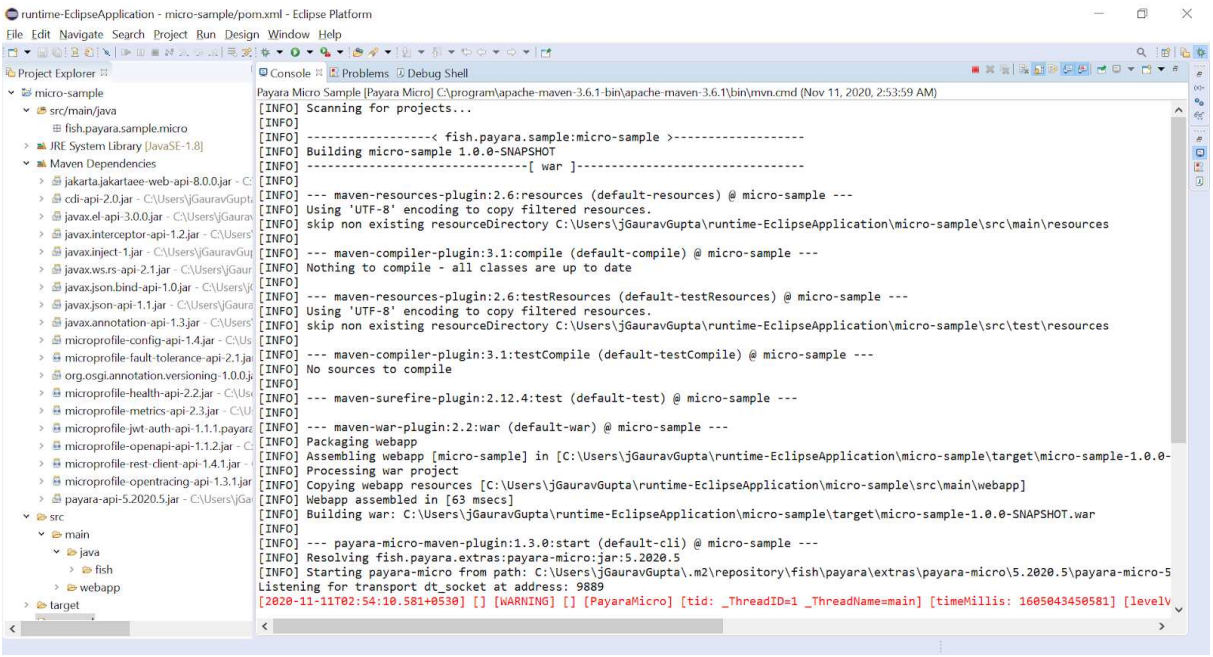
7. Add the configuration to **Run and Debug** menu and press the **Apply** button to save the configuration:

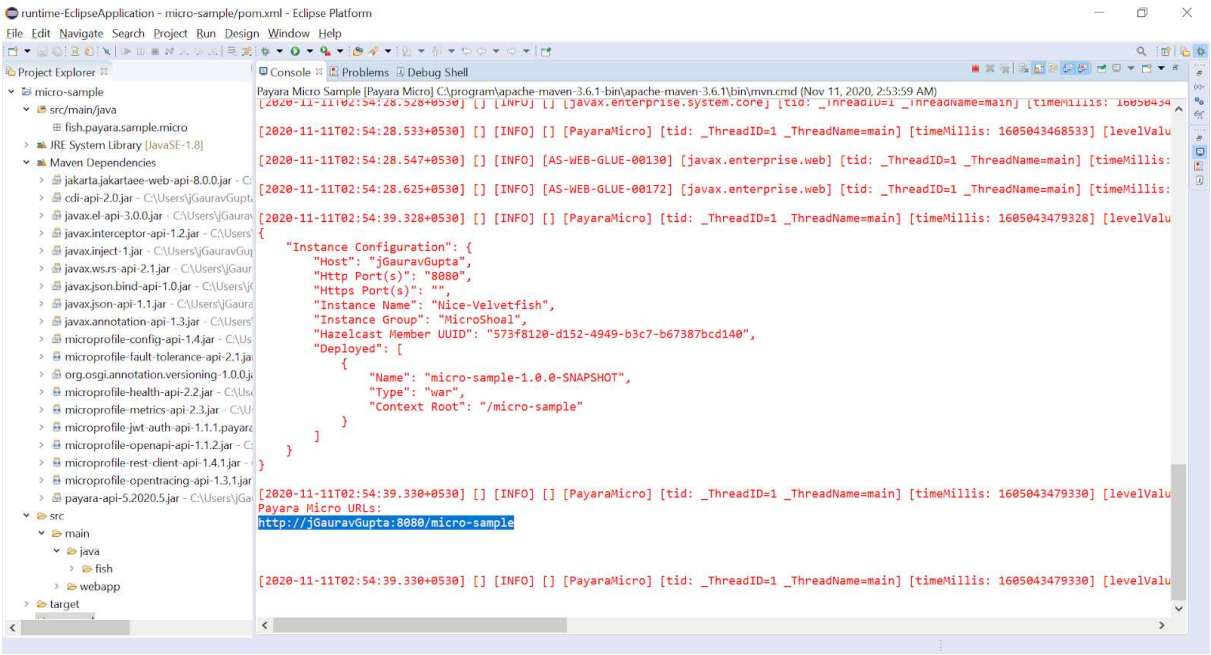


8. To start the Payara Micro instance, run the **Payara Micro Sample** configuration from the toolbar or main menu:



9. After running the Micro application, you may see the log in the OUTPUT window:

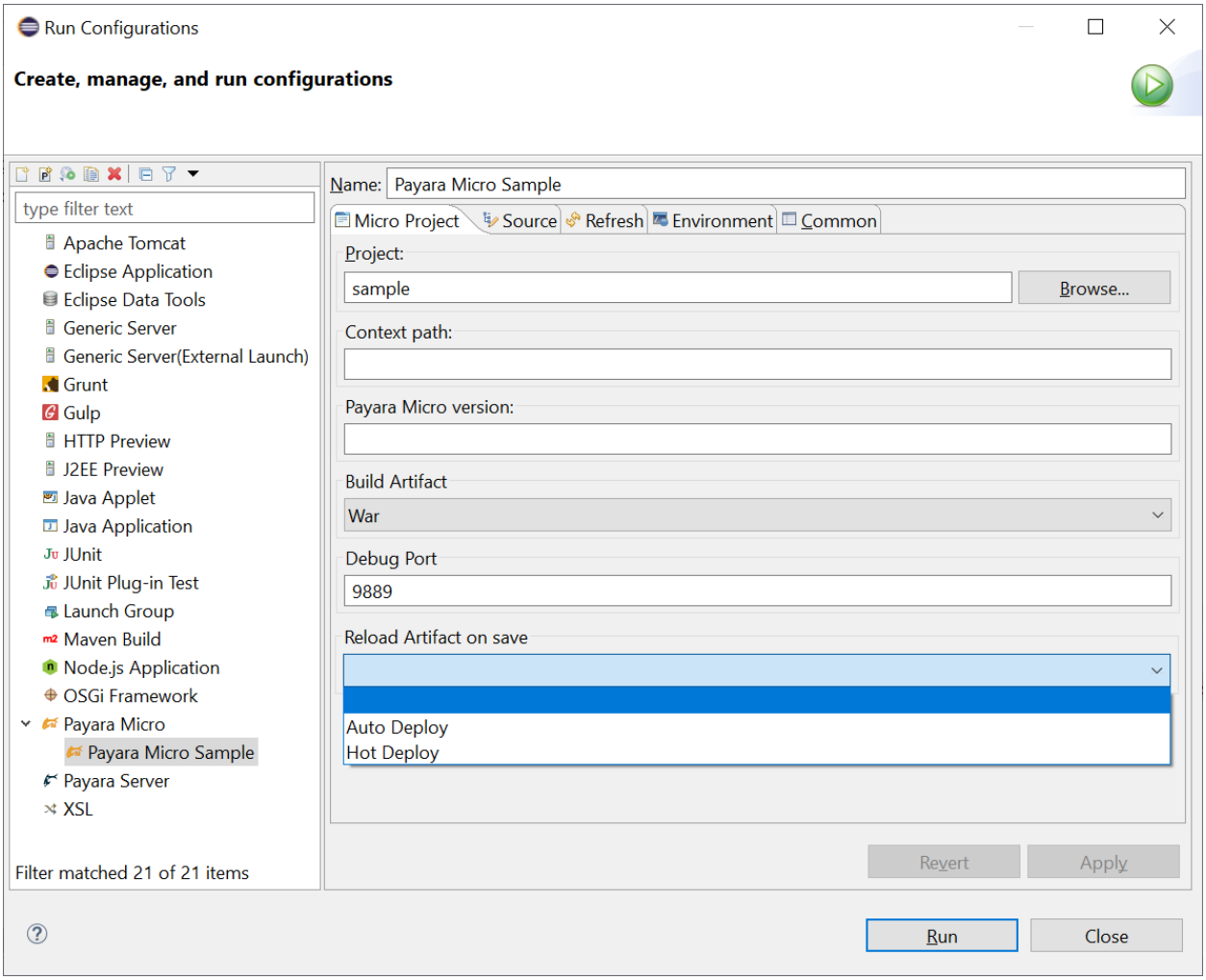




Similarly, to debug the Payara Micro application and attach the debugger to the application, instead of pressing the Run button, you can instead click **Debug** from similar locations and select the Payara Micro Sample configuration.

Hot Deploy and Auto Deploy Settings

To enable the **Auto Deploy** or **Hot Deploy** features for Payara Micro on the Eclipse IDE, modify your run configurations **Reload Artifact** action from the drop-down list:



See Also

"Hot Deploy and Auto Deploy"

Building Payara Tools Eclipse IDE Plugin

The Payara Tools Eclipse IDE Plugin source code is available in the Eclipse Plugin repository.

Prerequisites

Before building the Payara Tools Eclipse plugin, you will need to configure your local environment. Subsequent builds will not require these steps.

- **Clone the ecosystem-eclipse-plugin repository**: You must first clone the repository where the plugin source code is stored. Eclipse Plugin repository
- **Set the `JAVA_HOME` environment variable**: An environment variable named

JAVA_HOME must be created and point to a valid JDK 17 installation.

- **Install Apache Maven**: Apache Maven is the build tool for the Payara Tools Eclipse IDE Plugin and must be configured locally. Visit Apache Maven Installation to install Maven.

Build Payara Tools

To build Payara Tools for Eclipse, navigate to the root directory of the cloned repository and run the following command:

```
mvn clean install
```

This command will initiate the build process for the Payara Tools Eclipse plugin using Maven.

The build job will create both a zip and exploded-directory format. The zip format of this build can be found in the `[project_root]/releng/fish.payara.eclipse.update/target` directory, and the exploded-directory format can be found in the `[project_root]/releng/fish.payara.eclipse.update/target/repository` directory.

Install Payara Tools into Eclipse

After Payara Tools for Eclipse has been compiled successfully, you need to add it into the Eclipse IDE.

Adding Payara Tools to Eclipse

1. Navigate to Help Install New Software…
2. Click Add Archive…
3. Select the zip format in `[project_root]/releng/fish.payara.eclipse.update/target`
4. Select the Eclipse Payara Tools checkbox

5. Click Next Finish
6. Select Install Anyway
7. Restart Eclipse when prompted

[Installing Payara Tools] | *eclipse-plugin/Install Payara Tools.gif*

Removing Payara Tools from Eclipse

Removing Payara Tools from Eclipse after installing using the method above is the same as removing any plugin from Eclipse.

Removing Payara Tools from Eclipse

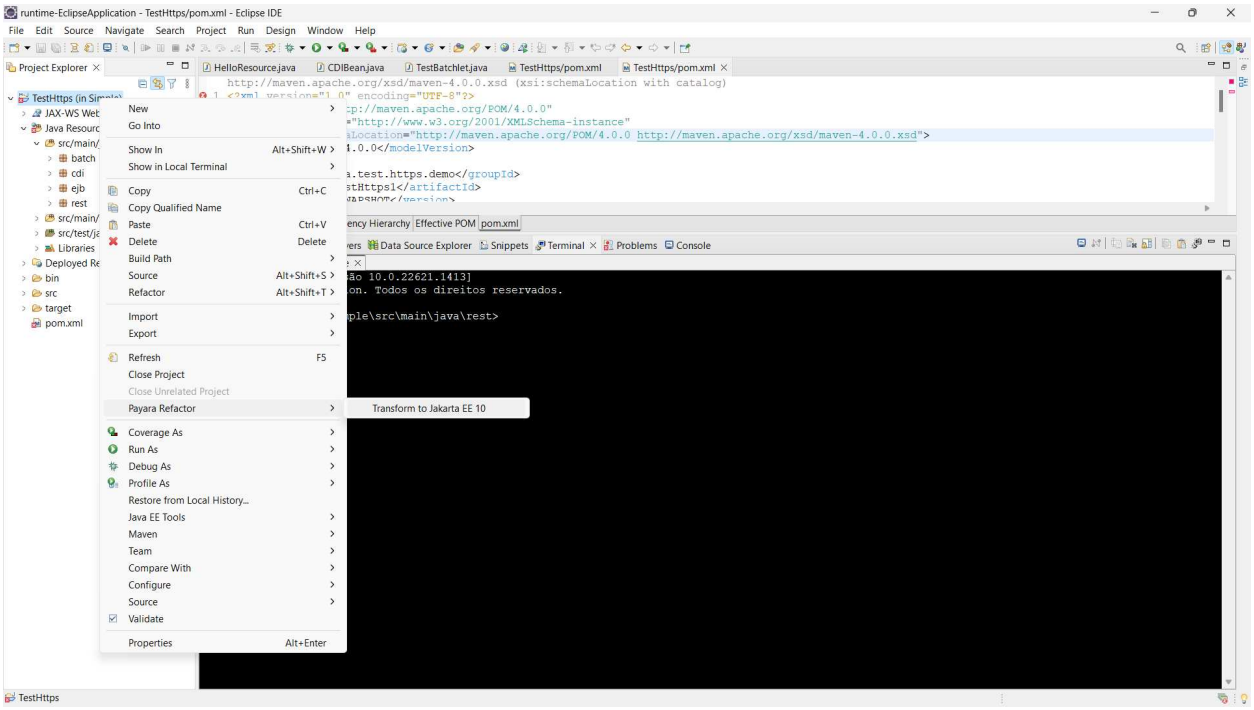
1. Navigate to Help Eclipse Marketplace
2. Open the Installed tab
3. Locate Payara Tools and click Uninstall

Transform Maven Projects or Files from Java EE 8 to Jakarta EE 10

Using the `Payara Tools plugin`, you can transform whole projects or single files from Java EE 8 to Jakarta EE 10.

Refactor a Java EE 8 Project to Jakarta EE 10

To refactor a Java EE 8 based **Maven** project right click on root project folder and choose **Payara RefactorTransform to Jakarta EE 10**:



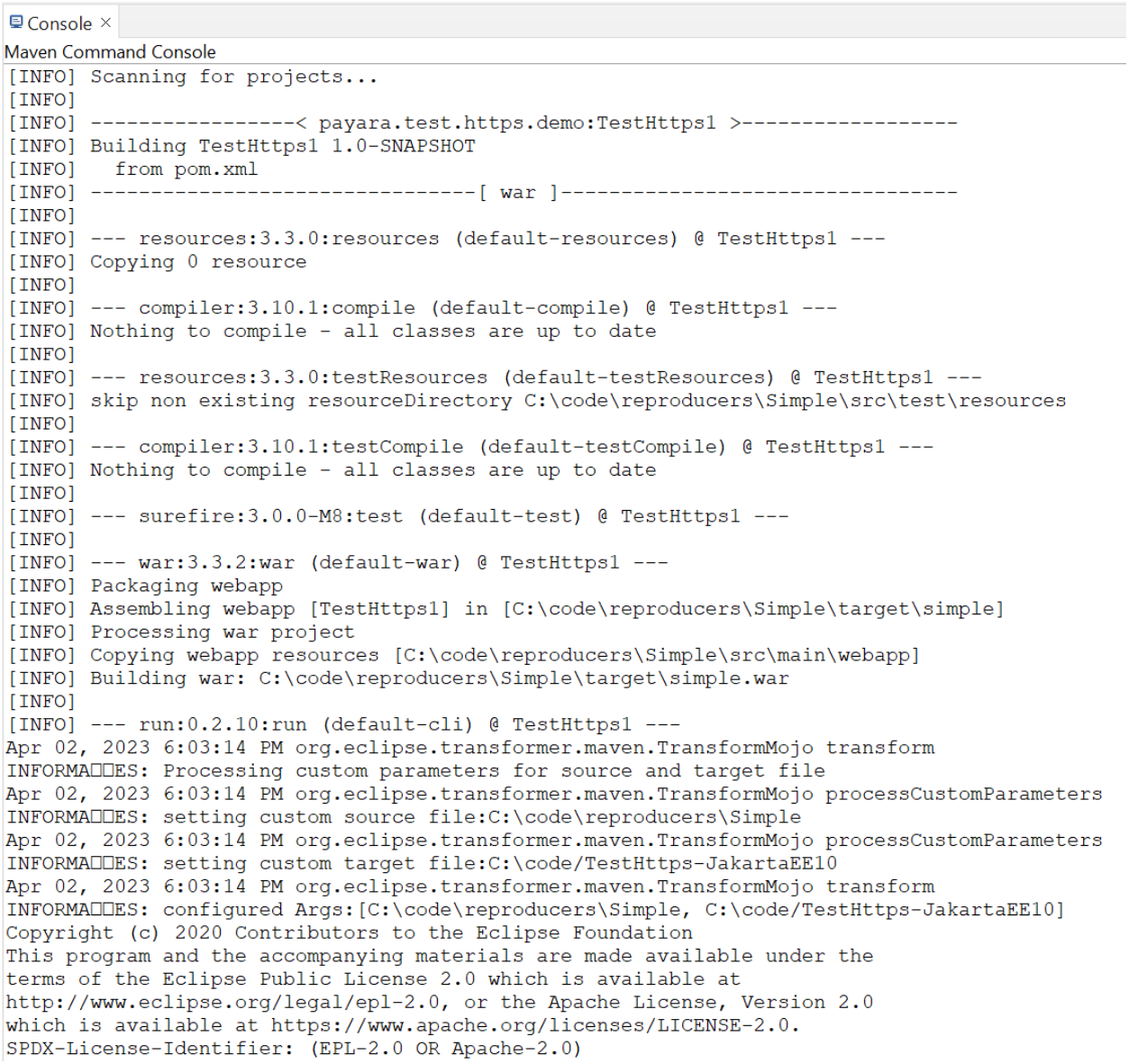
The plugin will open a dialog to choose the destination folder where the new project will be created:

[Choose Project Destination Folder] | *eclipse-plugin/transform-to-jakarta/choose-new-project-dest-folder.png*

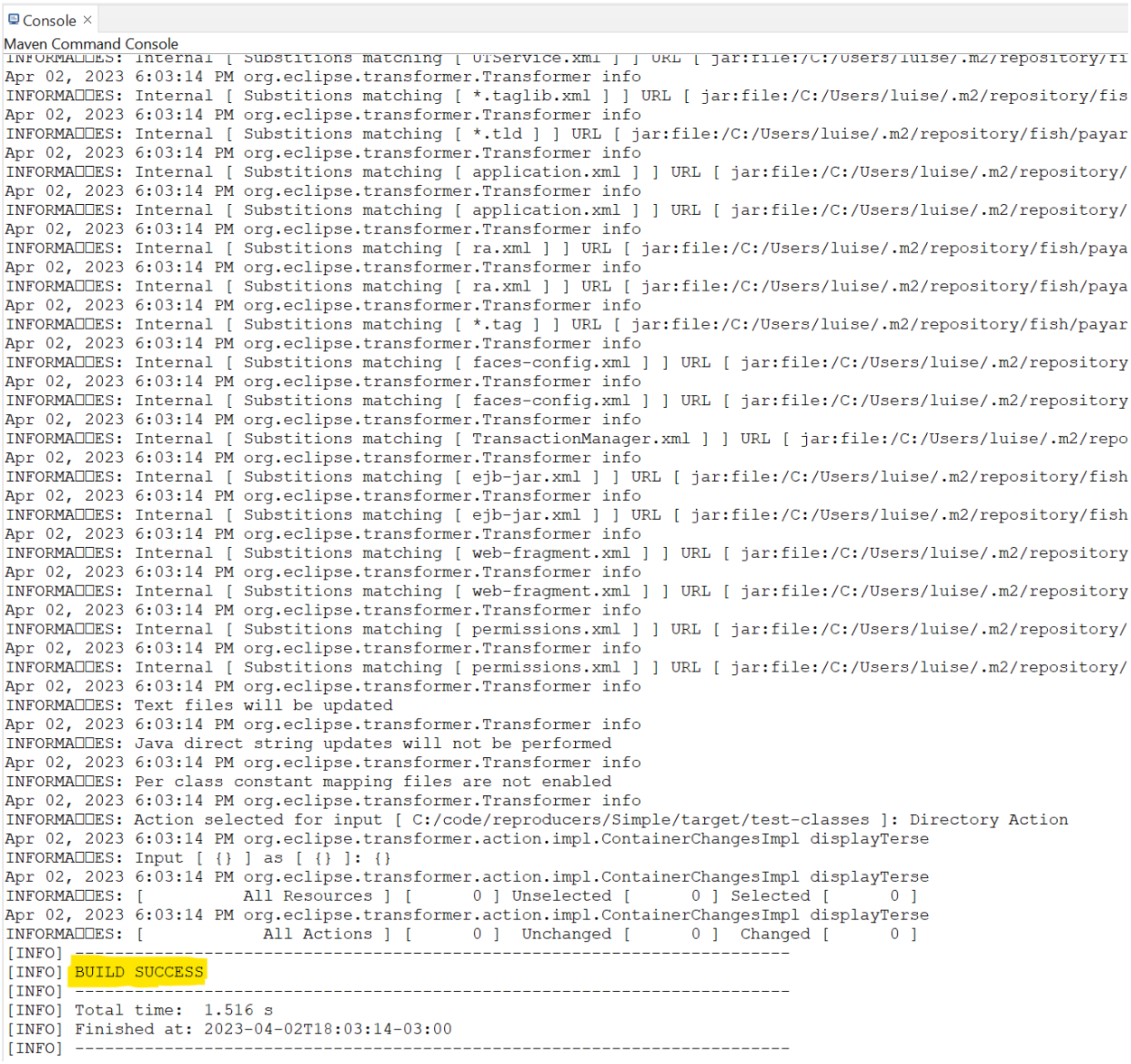
After choosing the `Destination Directory`, a confirmation message will appear indicating that the project was generated.

TIP | You can see the output of the generation process in the Eclipse console.

The Maven plugin that executes the transformation process will print its output to the IDE's console:



A maven `BUILD SUCCESS` message should be logged in the IDE's console.

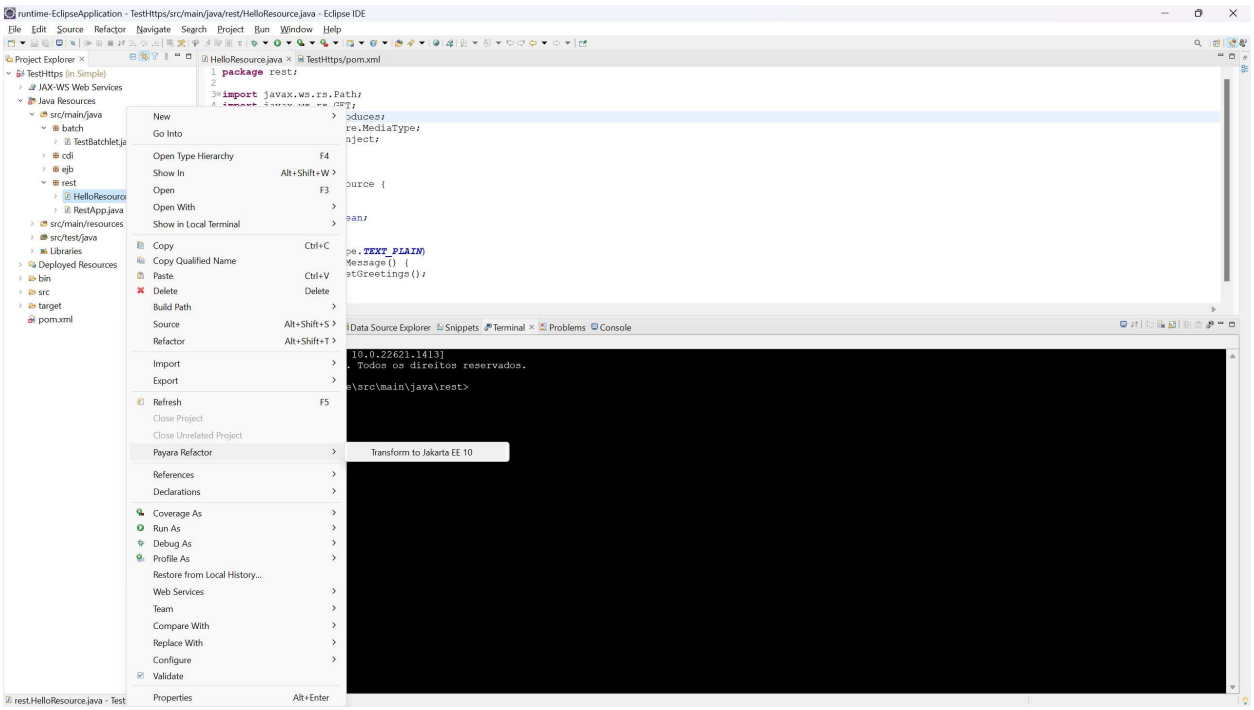


NOTE

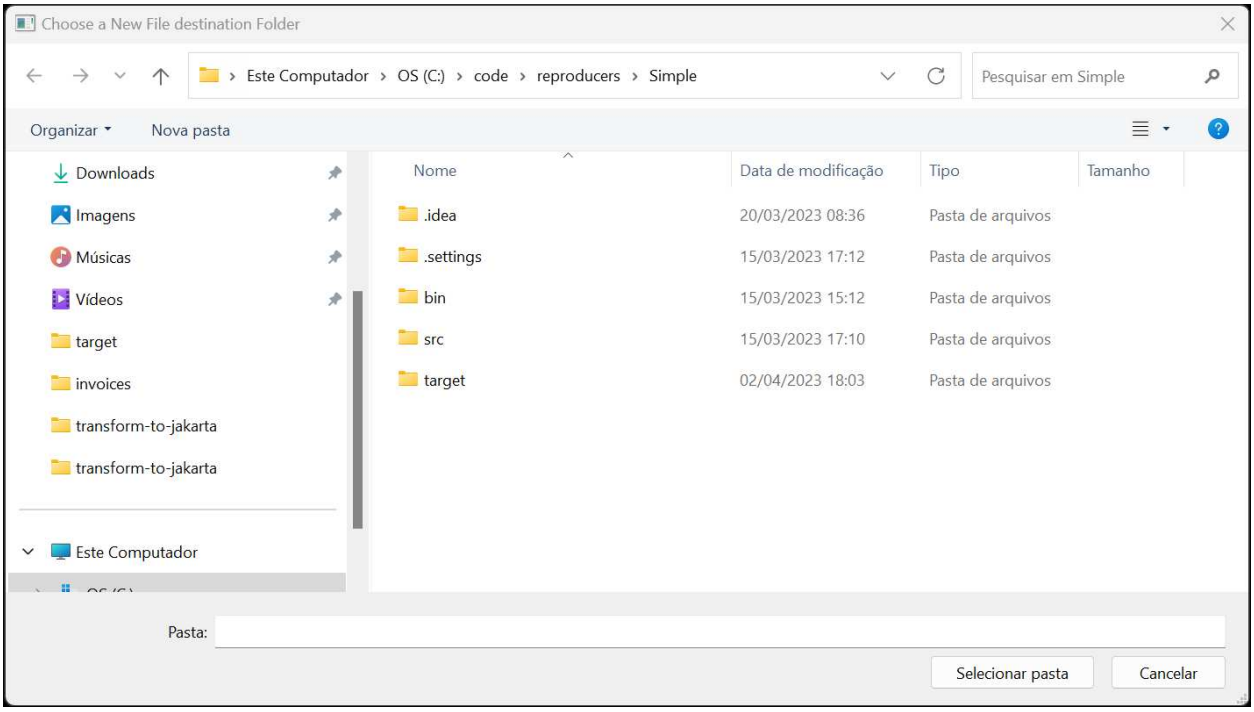
When the whole project is transformed, the plugin adds `JakartaEE10` at the end of the project name. In our example, `demo-JakartaEE10` project was created from `demo` project.

Transform a Single File

To refactor a `Java EE 8 file` right click on it and choose **Payara Refactor→Transform to Jakarta EE 10** option:



The plugin will open a dialog to choose the destination package where the new file will be created:



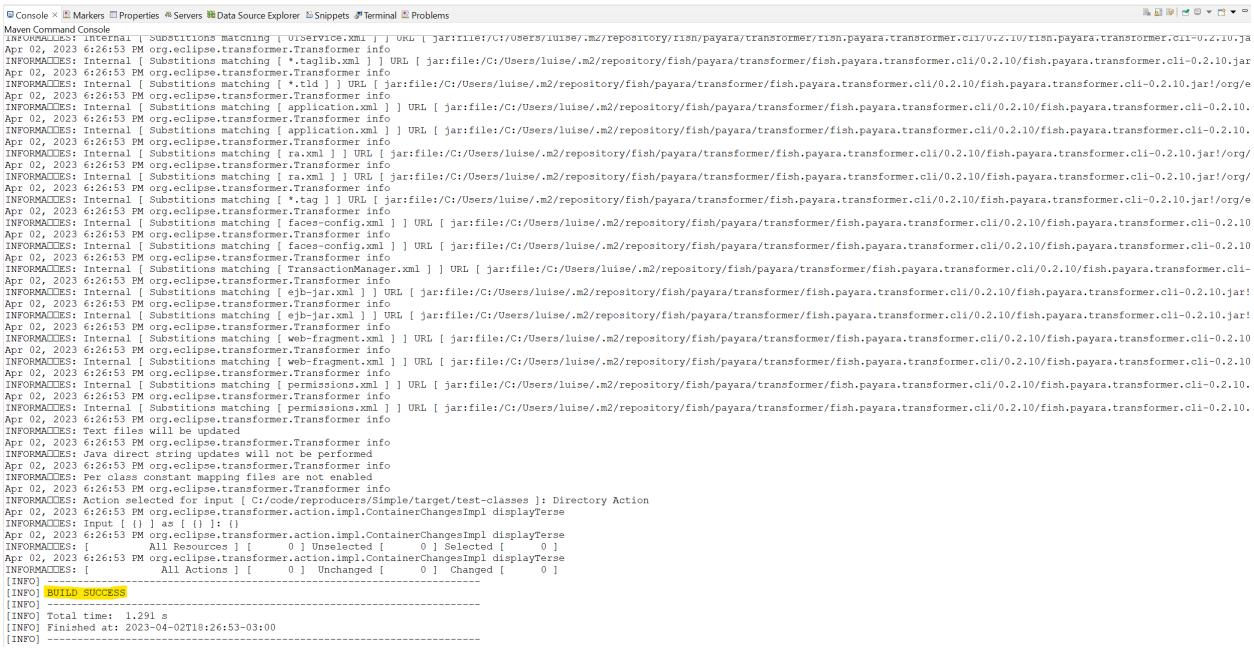
After choosing the `Destination Package`, a confirmation message will appear indicating that the file was generated.

TIP | You can see the output of the generation process in the Eclipse console.

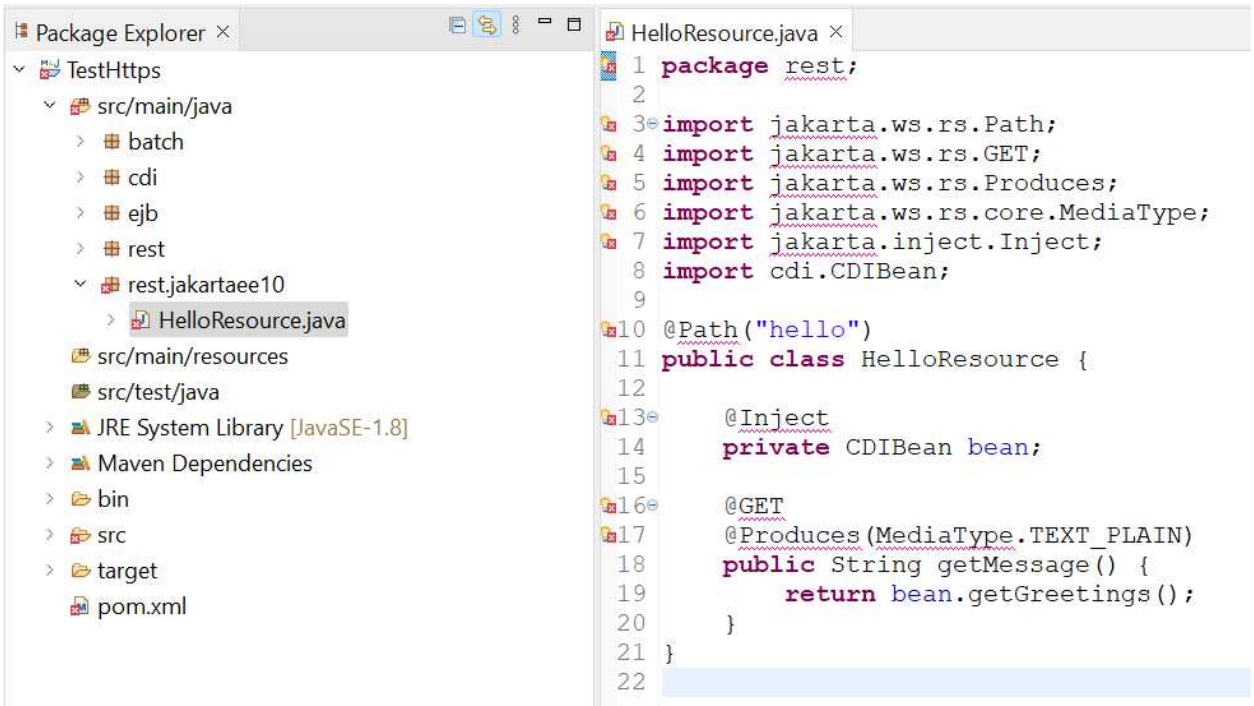
The output of the Maven plugin that executes the transformation code can be reviewed in the IDE's console:

```
Console × Problems Javadoc Declaration Terminal Servers
Maven Command Console
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------< payara.test.https.demo:TestHttps1 >------------------
[INFO] Building TestHttps1 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ war ]---------------------------------
[INFO]
[INFO] --- resources:3.3.0:resources (default-resources) @ TestHttps1 ---
[INFO] Copying 0 resource
[INFO]
[INFO] --- compiler:3.10.1:compile (default-compile) @ TestHttps1 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- resources:3.3.0:testResources (default-testResources) @ TestHttps1 ---
[INFO] skip non existing resourceDirectory C:\code\reproducers\Simple\src\test\resources
[INFO]
[INFO] --- compiler:3.10.1:testCompile (default-testCompile) @ TestHttps1 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- surefire:3.0.0-M8:test (default-test) @ TestHttps1 ---
[INFO]
[INFO] --- war:3.3.2:war (default-war) @ TestHttps1 ---
[INFO] Packaging webapp
[INFO] Assembling webapp [TestHttps1] in [C:\code\reproducers\Simple\target\simple]
[INFO] Processing war project
[INFO] Copying webapp resources [C:\code\reproducers\Simple\src\main\webapp]
[INFO] Building war: C:\code\reproducers\Simple\target\simple.war
[INFO]
[INFO] --- run:0.2.10:run (default-cli) @ TestHttps1 ---
Apr 02, 2023 6:52:59 PM org.eclipse.transformer.maven.TransformMojo transform
INFORMA□□ES: Processing custom parameters for source and target file
Apr 02, 2023 6:52:59 PM org.eclipse.transformer.maven.TransformMojo processCustomParameters
INFORMA□□ES: setting custom source file:C:\code\reproducers\Simple\src\main\java\rest\HelloResource.java
Apr 02, 2023 6:52:59 PM org.eclipse.transformer.maven.TransformMojo processCustomParameters
INFORMA□□ES: setting custom target file:C:\code\reproducers\Simple\src\main\java\rest/jakartaee10/HelloResource.java
Apr 02, 2023 6:52:59 PM org.eclipse.transformer.maven.TransformMojo transform
INFORMA□□ES: configured Args:[C:\code\reproducers\Simple\src\main\java\rest\HelloResource.java, C:\code\reproducers\Simple\src\main\java\rest/jakartaee10/HelloResource.java]
Copyright (c) 2020 Contributors to the Eclipse Foundation
This program and the accompanying materials are made available under the
terms of the Eclipse Public License 2.0 which is available at
http://www.eclipse.org/legal/epl-2.0, or the Apache License, Version 2.0
which is available at https://www.apache.org/licenses/LICENSE-2.0.
SPDX-License-Identifier: (EPL-2.0 OR Apache-2.0)
```

A `BUILD SUCCESS` message from the Maven plugin will be logged in the IDE's console.



The new file will be located in the chosen destination package.



NOTE

When a single file is transformed, a `jakartaee10` package containing the new file will be created in the destination package.

IntelliJ Plugin

Payara Intellij Tools

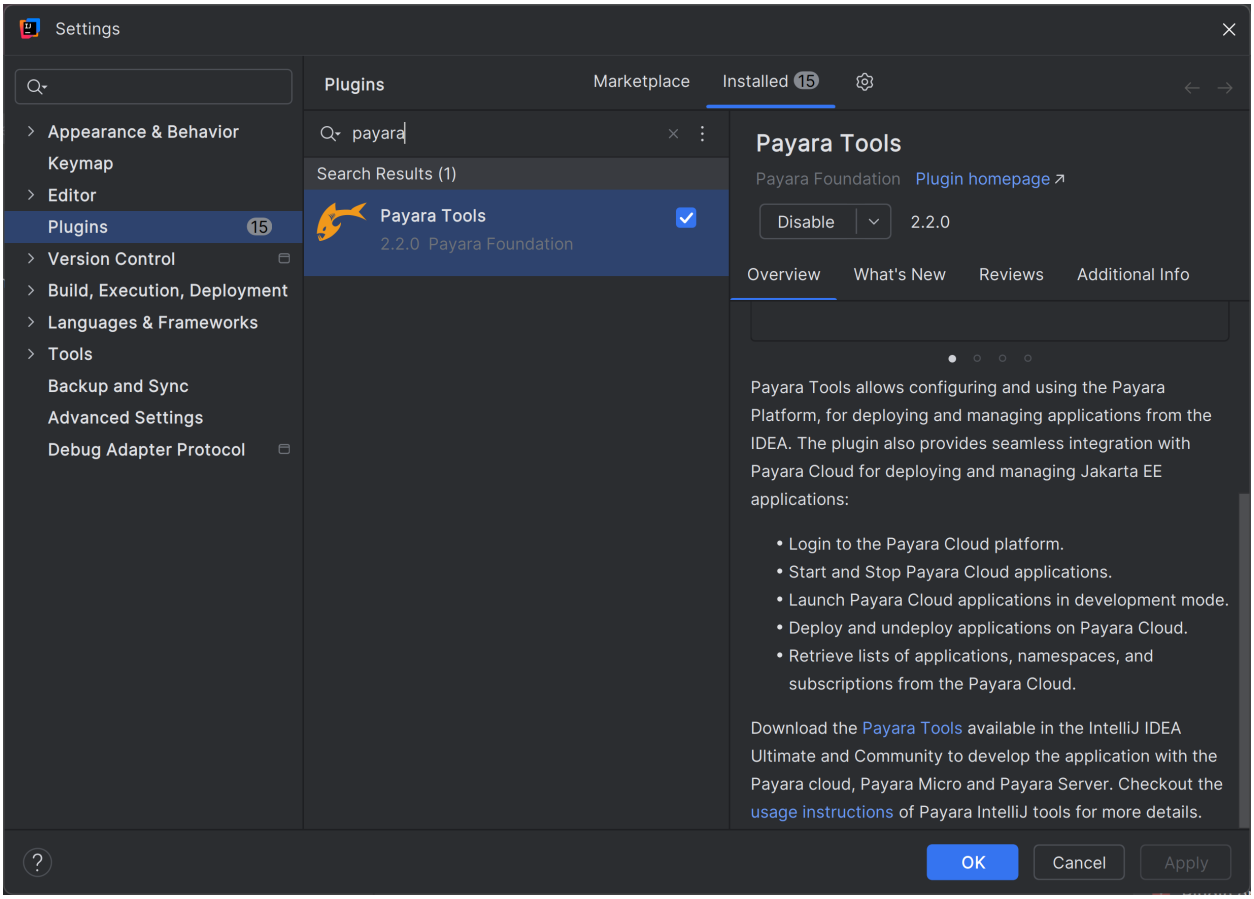
The Payara IntelliJ Ultimate Tools enables Payara Server, Payara Micro and Payara Cloud integration with IntelliJ IDEA Ultimate Edition. The Payara IntelliJ Ultimate Tools plugin is available in the JetBrains Plugins Marketplace.

If you are developing an application on IntelliJ IDEA Community Edition, you should install the Payara IntelliJ Community Tools plugin which enabled integration of Payara Micro with IntelliJ IDEA Community Edition. The Payara IntelliJ Community Tools Plugin is also available in the JetBrains Plugins Marketplace.

IMPORTANT	If you are using IntelliJ IDEA Ultimate, you should only install the `Payara IntelliJ Ultimate Tools` plugin. This contains support for both Payara Server and Payara Micro.

Installing Payara Tools

1. Open IntelliJ IDEA and navigate to **File > Settings > Plugins**
2. In the Plugins dialog, switch to the **Marketplace** tab
3. Search for **Payara**
4. Click the **Install** button for **Payara Tools**
5. Click **OK** to apply the changes and restart IntelliJ IDEA if prompted



Payara Intellij Tools Features

- "Payara Server" - The Payara IntelliJ Ultimate Tools enables developers to easily configure Payara Server and manage applications from IntelliJ IDEA.
- "Payara Micro" - The Payara IntelliJ Ultimate Tools provide support to create, run and debug Payara Micro applications from IntelliJ IDEA. Payara Micro is the microservices-ready version of the Payara Server.

Payara Server Tools in Intellij IDEA

IMPORTANT	This is only available in the `Payara IntelliJ Ultimate Tools` plugin which is exclusive to IntelliJ IDEA Ultimate

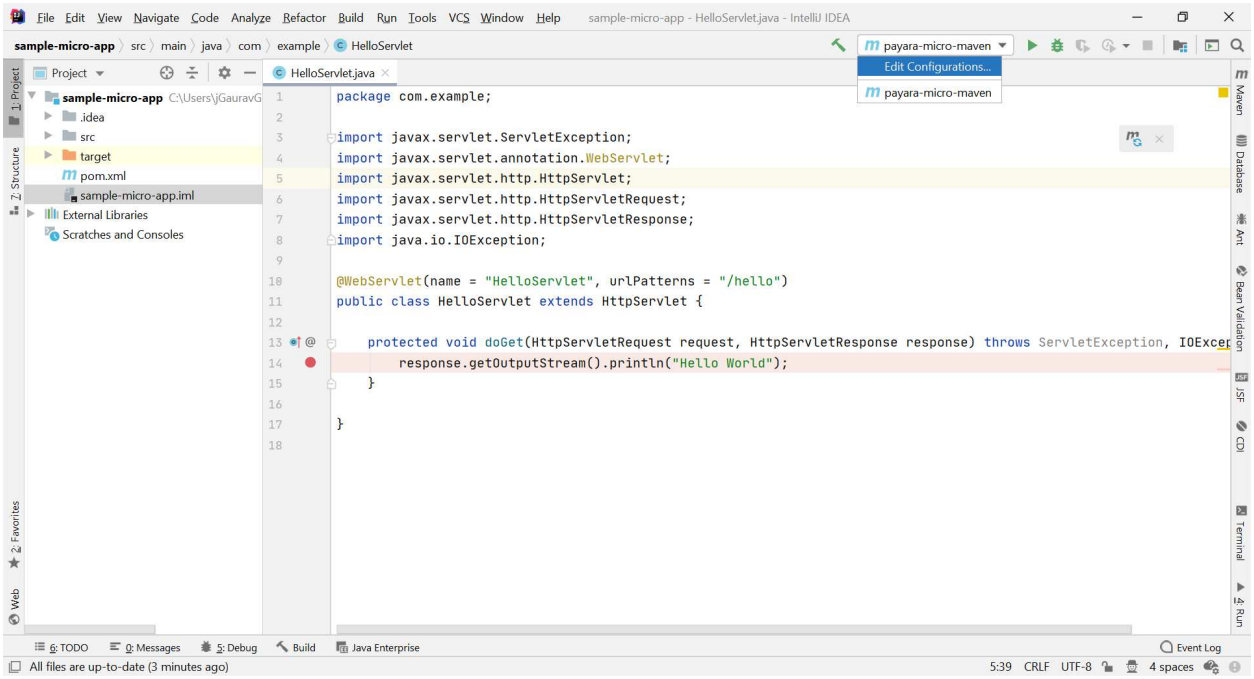
Using the Payara IntelliJ Ultimate Tools plugin, you can integrate Payara Server within the IntelliJ IDEA and perform the following tasks:

- Start and stop the Payara Server from within the IDE

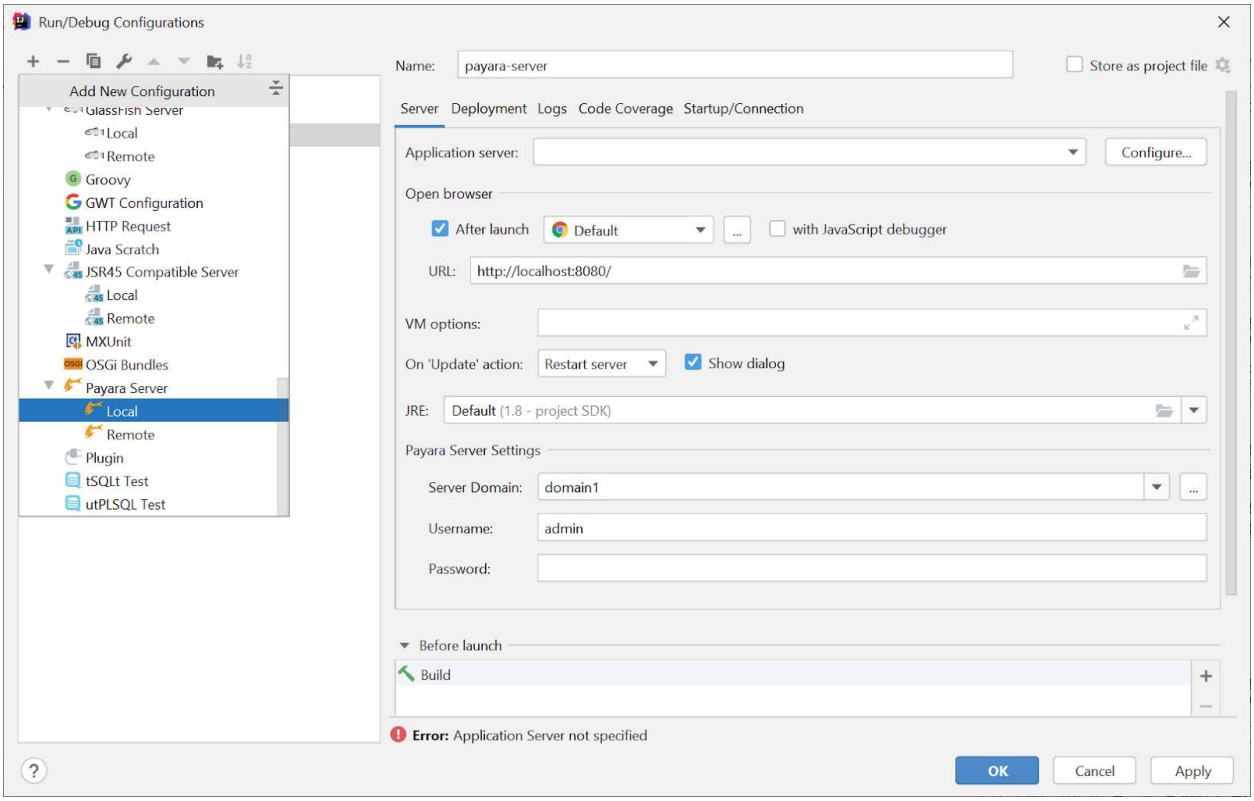
- Define the Java runtime to be used by Payara Server
- Deploy an application from within the IDE

Managing Payara Server from the Intellij IDE

To manage Payara Server from the IntelliJ idea, first create the new **Payara Server configuration** by selecting the **Edit Configuration** option:



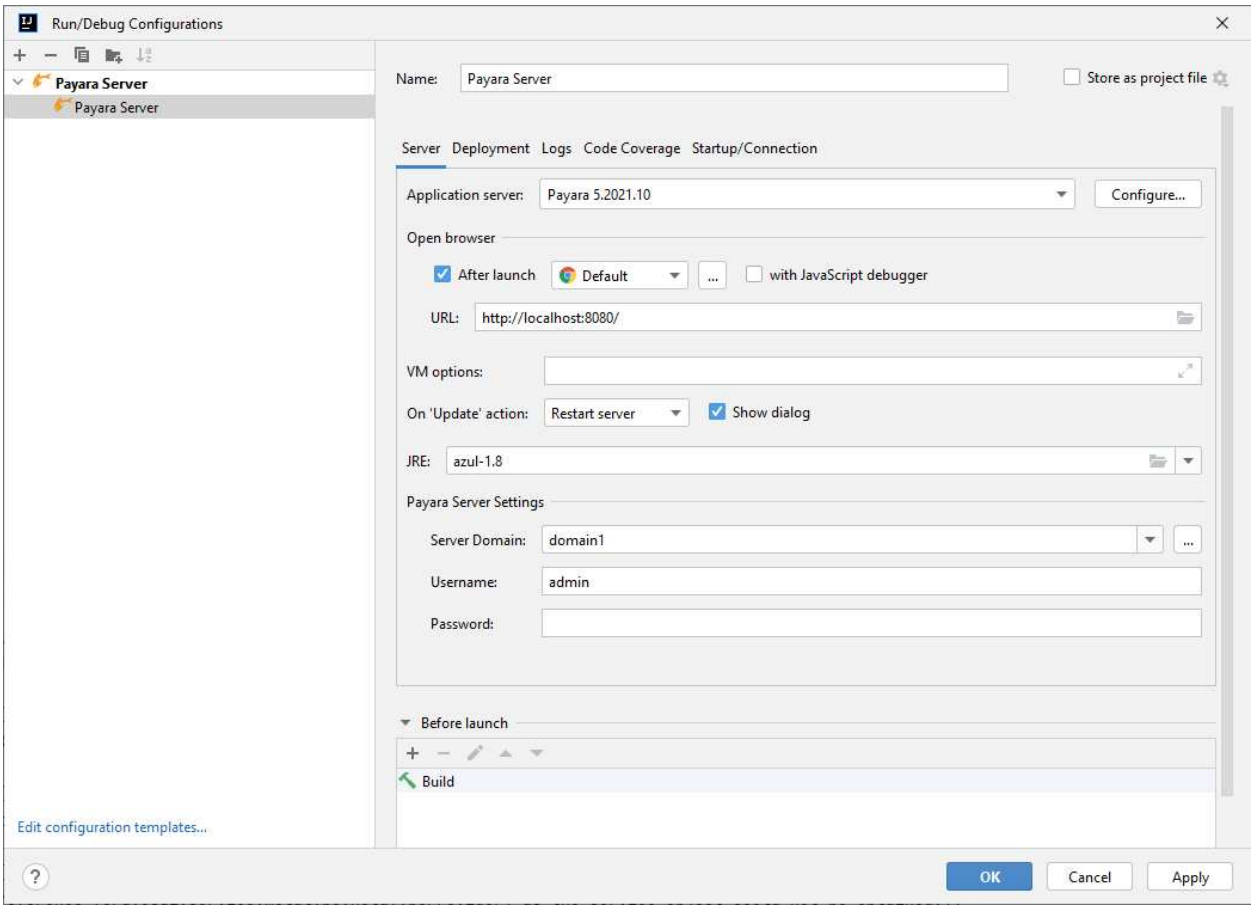
In the **Run/Debug Configurations** dialog, click **+**, expand the **Payara Server** node, and select **Local**:



To add the Payara Server to the configuration, click the **Configure…** button and set it to the Payara Server's installation path you would like to add to the IDE.

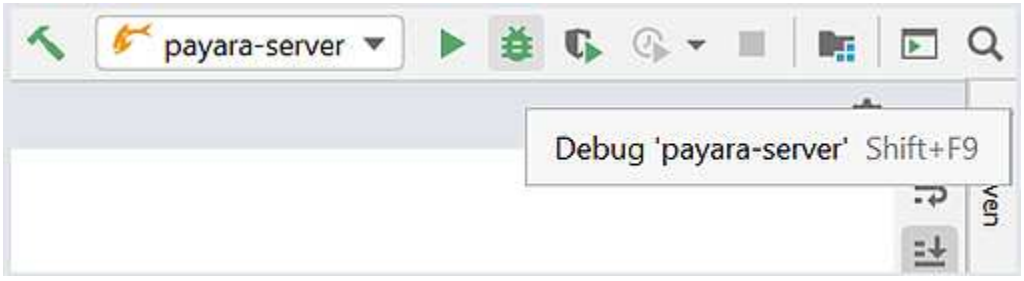
Specify the path to the JRE or JDK version you would like Payara Server to run on.

Finally, fix the debugger related warning by pressing the **Fix** button, which updates the debugger settings in the domains.xml of the selected Payara Server's domain.



For now, ignore the `No artifacts marked for deployment` warning and click **OK** to save the configuration

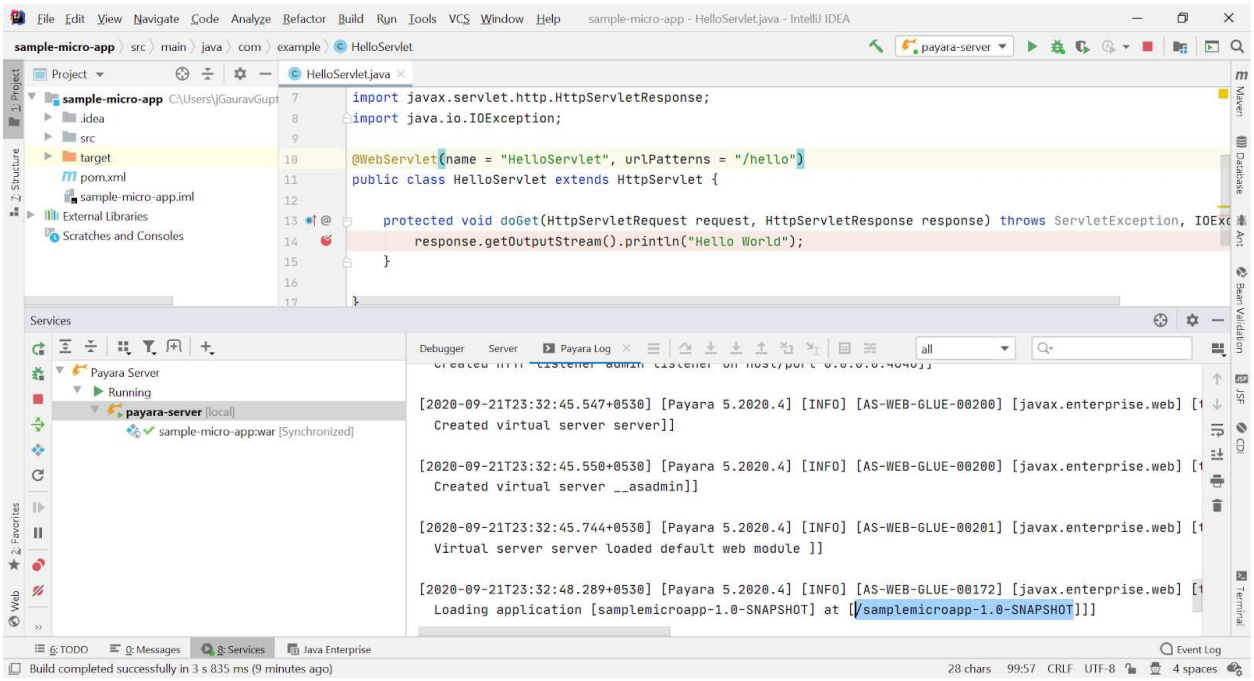
Click the **Run or Debug** button from top-right panel to start Payara Server, or start Payara Server in debug mode, which would allow you to debug deployed applications



You can see the server log in the **Run > Payara Log** window.

TIP

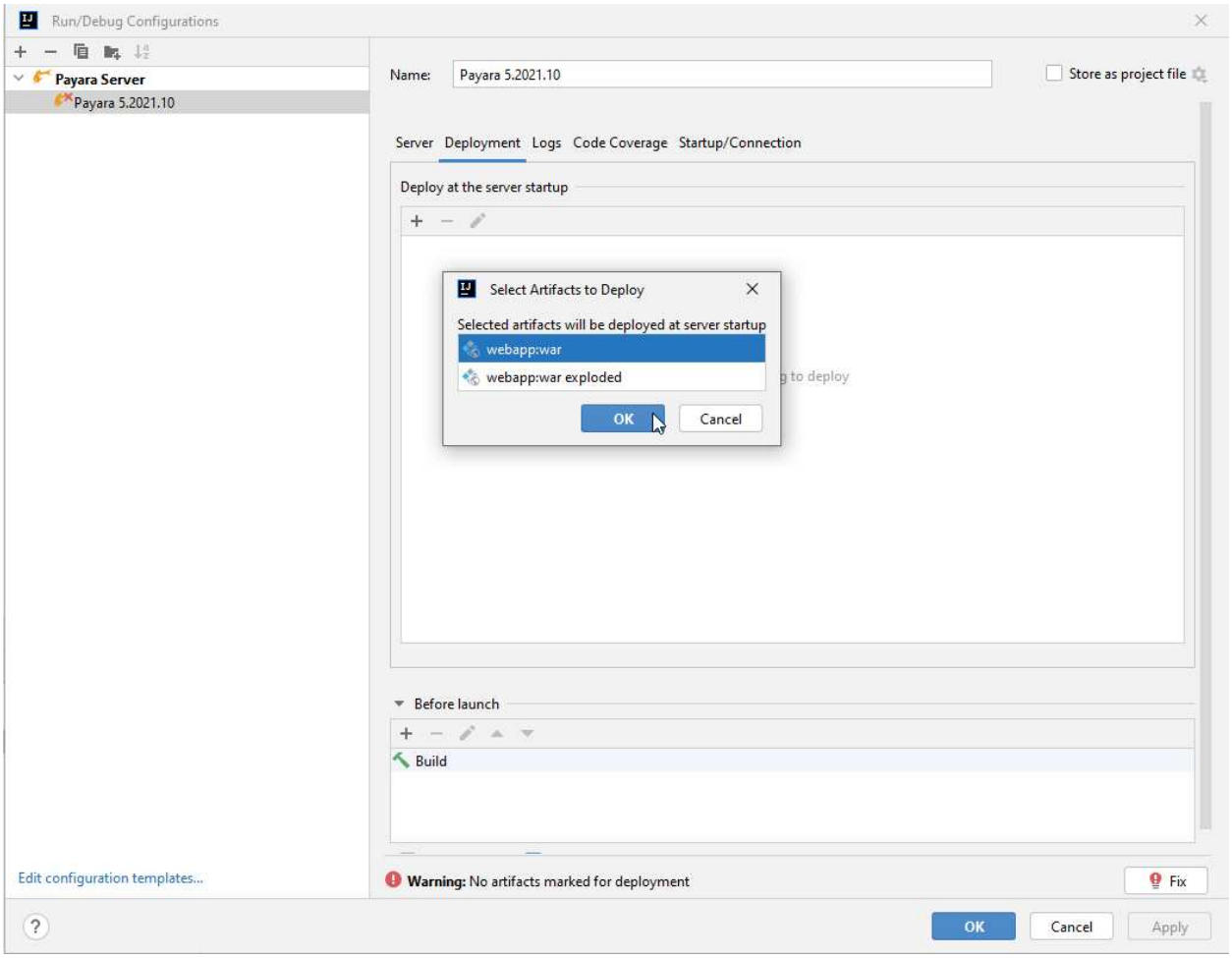
You can filter logs by level with the dropdown menu on the right of the `Payara Log`



Deploying an Application to Payara Server from Intellij

To deploy an application to Payara Server you must further edit the run configuration you are using to run Payara Server from IntelliJ.

1. Select the Run Configurations drop down menu
2. Click on **Edit Configurations**
3. Choose the Payara Server configuration to which you want to deploy the application.
4. In the configuration settings, navigate to the **Deployment** section and click on the **+** button.
5. Select **Artifact…** then choose the non-exploded format of your application's name.
6. Click **OK**



After application deployed to the Payara Server, IntelliJ starts a web browser and opens the application context root in the web browser.

Deploy Applications in a Local Docker Container

The Payara Server tools for IntelliJ IDEA include Docker integration, making it seamless to deploy exploded applications to a Payara Server instance running within a local Docker container.

1. Run the Payara Server Docker container using the following command:

```
docker run -p 8080:8080 -p 4848:4848 -p 9009:9009 -v HOST_PATH:CONTAINER_PATH
payara/server-full:{release-id}
```

   Replace `HOST_PATH` with the actual host path and `CONTAINER_PATH` with the desired container path that must be the exploded application path or parent path of application.

The `HOST_PATH` points to the location on your host machine, and `CONTAINER_PATH` is the corresponding location inside the container.

2. Open Intellij IDEA Ultimate and first create the new **Payara Server configuration** by selecting the **Edit Configuration** option:
3. In the **Run/Debug Configurations** dialog, click `+`, expand the **Payara Server** node, and select **Remote**.
4. To add the Payara Server to the configuration, click the **Configure…** button and set it to the Payara Server's installation path with the same version as the one running in the Docker container.
5. For Docker integration, choose the "Docker Volume" option and provide the `HOST_PATH` (e.g., `D:/test`) and `CONTAINER_PATH` (e.g., `/mnt/test`) mapping details.

   This mapping should align with the exploded application path or parent path of exploded application.

6. Navigate to the **Deployment** section and click on the `+` button to select **Artifact…**.
7. Click **OK**
8. Click the **Run or Debug** button from top-right panel to deploy the application to the registered Payara Server docker instance.

Deploy Applications in a WSL Environment

Another noteworthy feature is the integration of the Payara Server Tools with Windows Subsystem for Linux (WSL).

This integration empowers developers to manage and deploy exploded applications to a Payara Server instance running within WSL.

Here's how you can make use of this feature:

1. Start the Payara Server instance within WSL:

a. Run the following command to start the server:

```
./asadmin start-domain
```

b. Set a password for the admin user:

```
./asadmin change-admin-password
```

c. Enable secure admin:

```
./asadmin enable-secure-admin
```

d. Restart the Payara Server domain:

```
./asadmin restart-domain
```

2. Register the WSL-based Payara Server instance in the IntelliJ IDEA:

a. Open Intellij IDEA Ultimate and first create the new **Payara Server configuration** by selecting the **Edit Configuration** option:

b. In the **Run/Debug Configurations** dialog, click `+`, expand the **Payara Server** node, and select **Remote**.

c. To add the Payara Server to the configuration, click the **Configure…** button and set it to the Payara Server's installation path running within WSL on your Windows machine.

d. For WSL integration, choose the "WSL" option and make sure **Use SSL Connection** option is checked.

e. Enter the host (fetched via the command `hostname -I` in WSL instance) and the necessary credentials.

f. Navigate to the **Deployment** section and click on the **+** button to select **Artifact…**.

g. Click **OK**

h. Click the **Run or Debug** button from top-right panel to deploy the application to the

registered Payara Server WSL environment.

Payara Server Maven Plugin Tools in Intellij IDEA

Using the Payara Tools and Payara Server Maven Plugin, you can integrate Payara Server within the IntelliJ IDEA to streamline development workflows. This integration allows you to:

- Start and stop the Payara Server from within the IDE
- Define the Java runtime to be used by Payara Server
- Deploy an application from within the IDE
- Configure pre-launch Maven tasks for automatic build and deployment

Managing Payara Server Using Maven Plugin from Intellij IDEA

1. To manage Payara Server using the Maven plugin from IntelliJ IDEA, first create the new **Payara Server configuration** by selecting the **Edit Configuration** option: image::intellij-plugin/payara-server-maven-plugin/create-configuration.png[Create Configuration,width=100%]
2. In the **Run/Debug Configurations** dialog, click **+**, select the **Payara Server**: image::intellij-plugin/payara-server-maven-plugin/create-server-configuration.png[Create Server configuration,width=100%]
3. In the newly created configuration:
   a. Set the goal to `dev` to deploy the application in development mode using the Payara Server Maven Plugin or `start` to start the server without dev mode.
   b. Specify the JDK or JRE path that Payara Server should use.
4. Optionally, configure the application's context path or deployment format (e.g., exploded). image::intellij-plugin/payara-server-maven-plugin/app-configuration.png[App configuration,width=100%]
5. To add the Payara Server to the configuration, click the **Payara Server Settings > Payara Home** and set it to the Payara Server's installation path you would like to add to the IDE. image::intellij-plugin/payara-server-maven-plugin/payara-

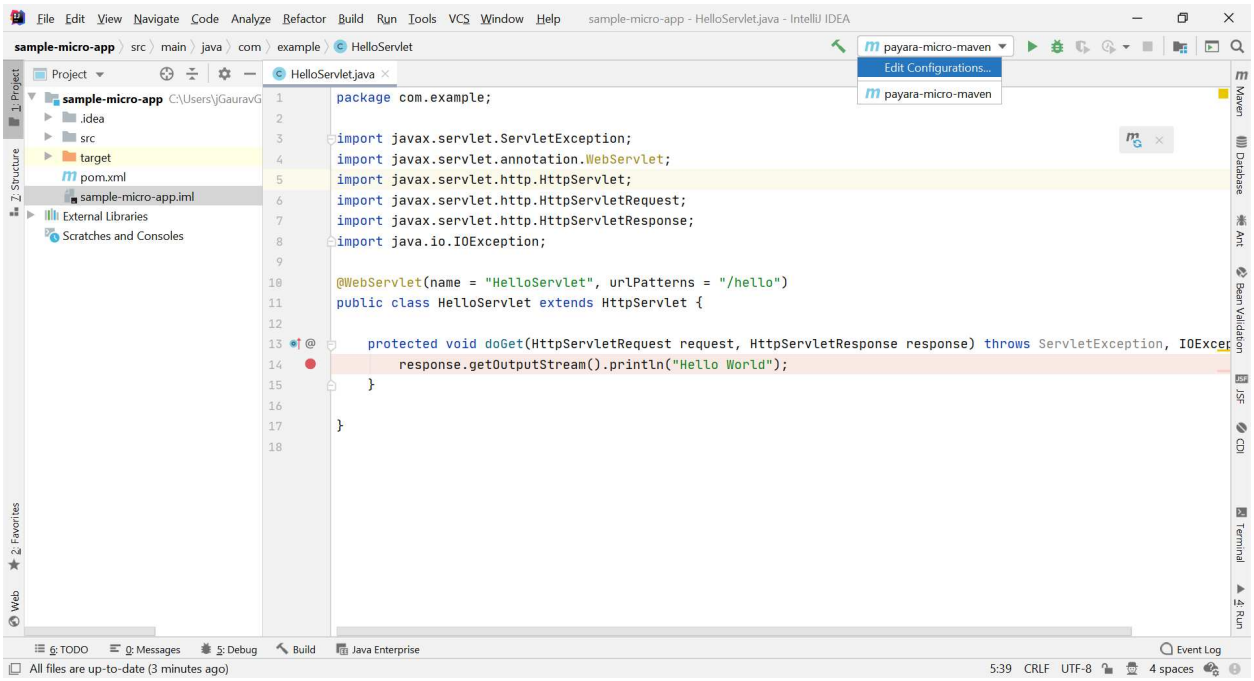
configuration.png[Payara configuration,width=100%]

6. Configure environment variables and system properties required by the server and Maven plugin. image::intellij-plugin/payara-server-maven-plugin/env-server-properties.png[Env and Server properties,width=100%]
7. Define additional Maven properties used by the Payara Server Maven Plugin during the build and deployment process. image::intellij-plugin/payara-server-maven-plugin/maven-properties.png[Maven properties,width=100%]
8. Specify Maven tasks to run before launching the server. image::intellij-plugin/payara-server-maven-plugin/before-launch-task.png[Before Launch Task,width=100%]
9. Click the **Run or Debug** button from top-right panel to start Payara Server, or start Payara Server in debug mode, which would allow you to debug deployed applications image::intellij-plugin/payara-server-maven-plugin/run-server.png[Start the Payara Server,width=100%]
10. You can see the server log in the **application build console** window. image::intellij-plugin/payara-server-maven-plugin/server-log.png[Payara Server log,width=100%]
11. After application deployed to the Payara Server, IntelliJ starts a web browser and opens the application context root in the web browser.

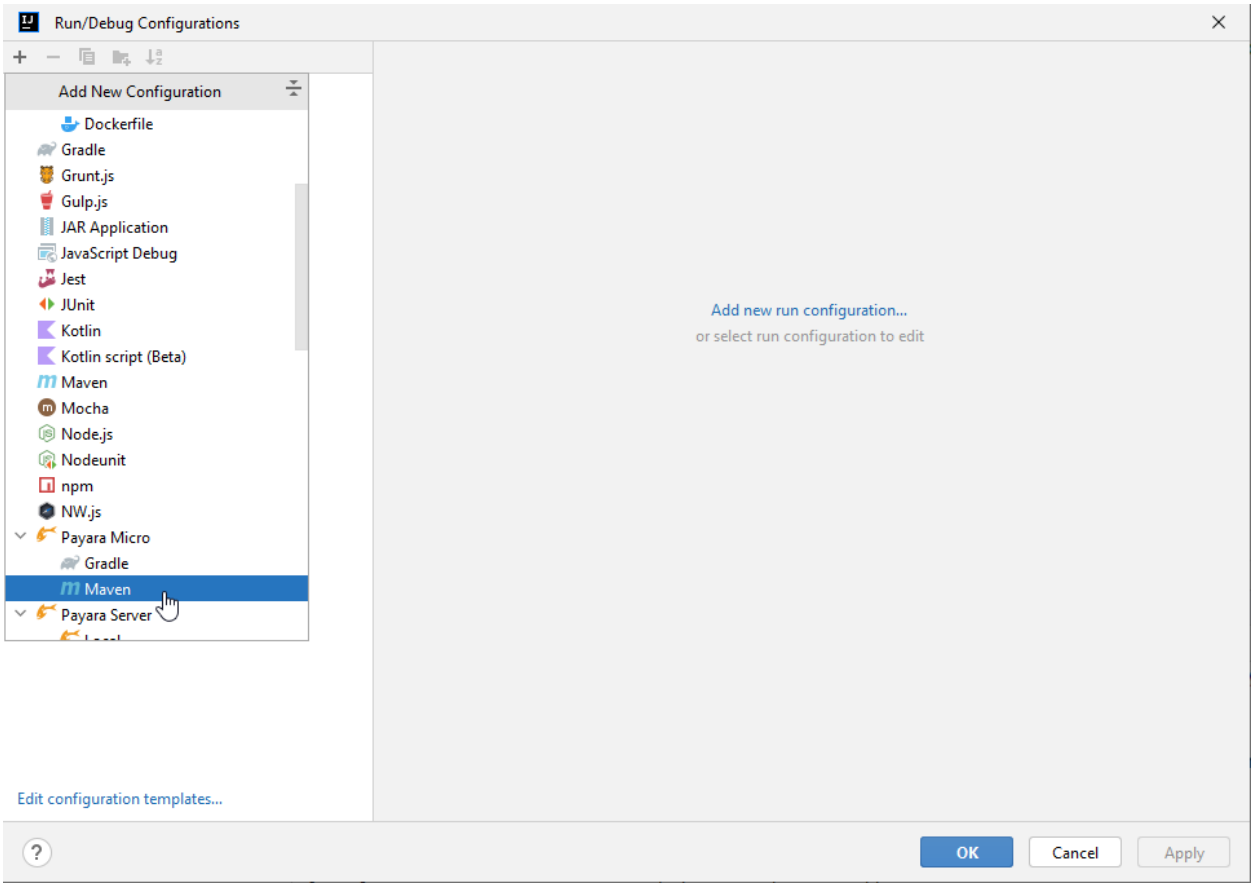
Payara Micro Tools in Intellij IDEA

Run Payara Micro from Intellij

To run Payara Micro from IntelliJ, you need to first configure Payara Micro as a run configuration.



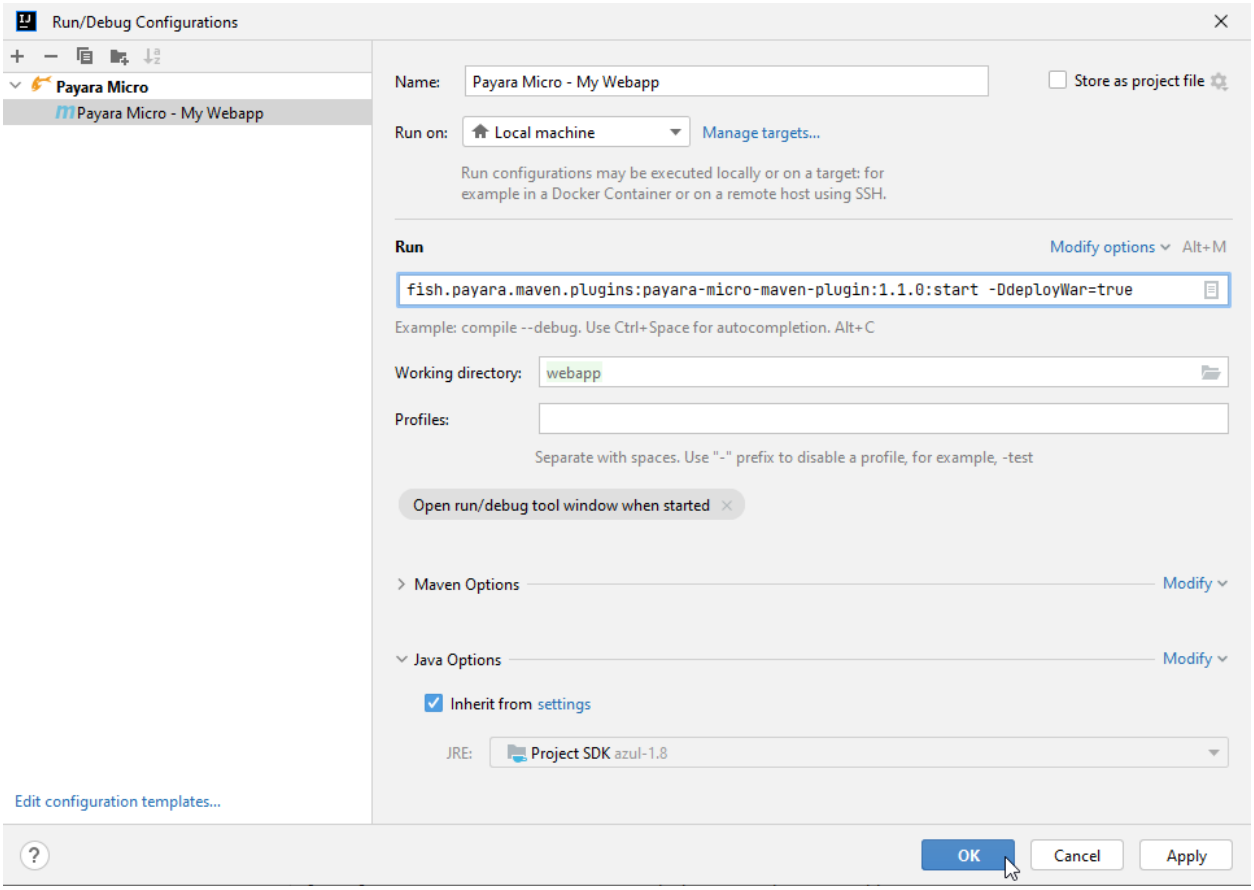
In the **Run/Debug Configurations** dialog, click **+**, expand the **Payara Micro** node, and select **Maven** or **Gradle**:



This run configuration will be configured by default to deploy the project you are

currently working on. If you wish to modify which application is deployed, edit the `Working Directory` option. Otherwise the default settings are correct, and click **OK**.

IMPORTANT	Due to a known issue within the IntelliJ IDEA Payara plugin, you must modify the default `Run` option by removing the leading and trailing quotation marks, `"`.

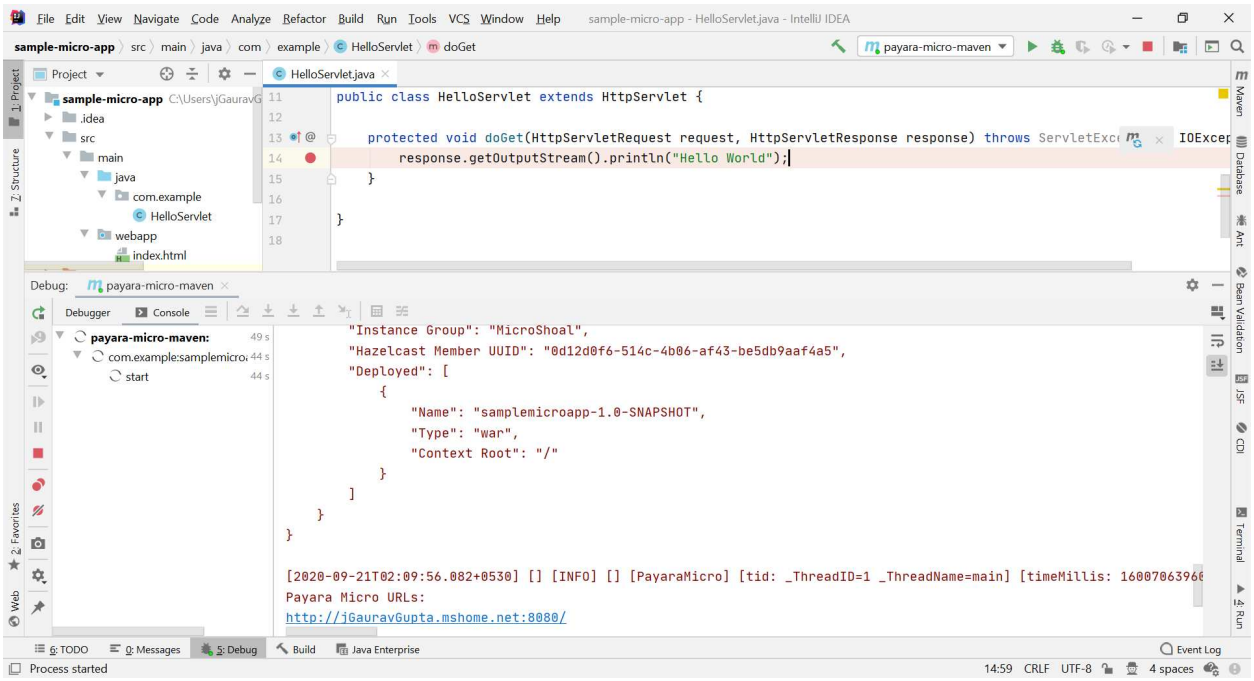


To run Payara Micro with your application compiled and deployed, just click **Run**. You can view the Payara Micro log in the **Run** window then select your run configurations name.

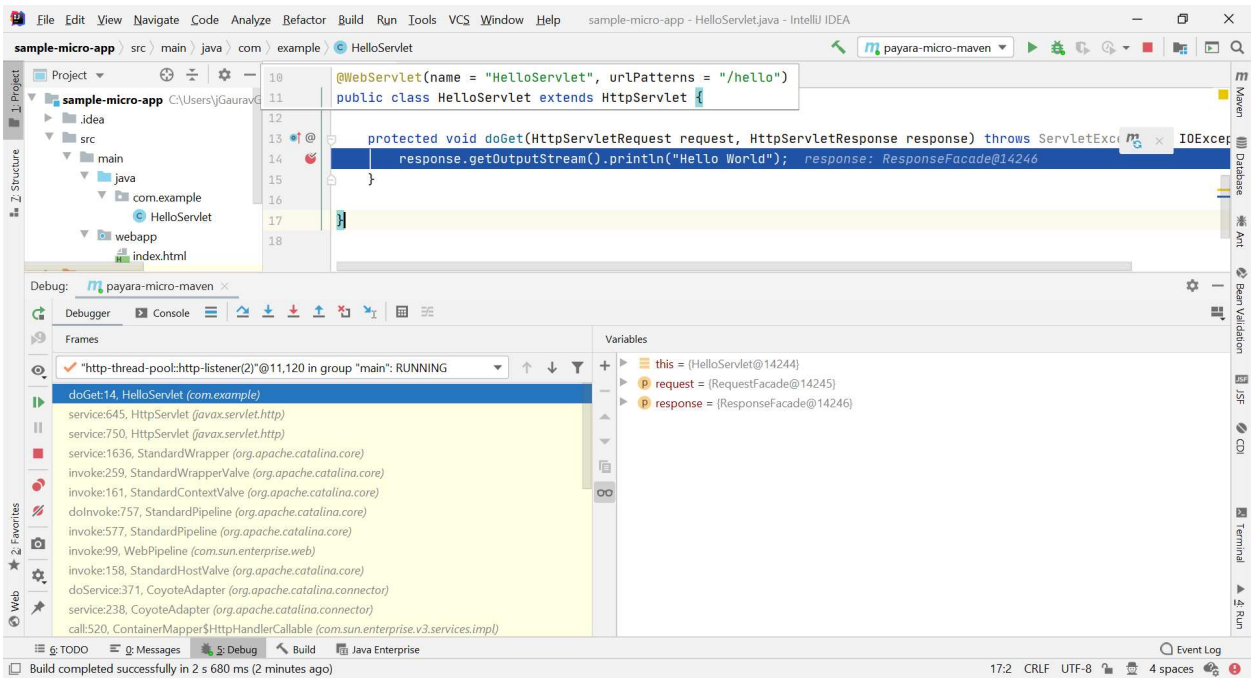
Debugging the Payara Micro Application

Instead of pressing the **Run** button, you can instead click **Debug** to automatically start the micro instance and attach the debugger to the application

After debugging the Payara Micro application, you can see the Payara Micro log in the **Debug** window then select your run configurations name.



After the debugger session starts and the application hits the breakpoint, the code execution is paused at the line where the breakpoint was placed, showing a list of running threads and the stacktrace:



Building Payara Intellij Tools

The process of building and testing the IntelliJ IDEA IDE Plugin, is the same for both the Ultimate and Community versions of the plugin. After cloning the appropriate repository,

the following steps can be applied to either plugin.

The Payara IntelliJ Community Tools source code is available in the Ecosystem IntelliJ Community repository.

The Payara IntelliJ Ultimate Tools source code is available in the Ecosystem IntelliJ Ultimate repository.

Prerequisites

Before building the Payara IntelliJ Tools, you will need to configure your local environment. Subsequent builds will not require these steps.

- **Clone the relevant repository**: You must first clone the appropriate repository for the IntelliJ IDEA plugin you are trying to build.
- **JDK 11**: JDK 11 is required to build and test the IntelliJ IDEA plugin.
- **Install Gradle**: Gradle is the build tool for the Payara IntelliJ Tools. You can either download and configure Gradle locally or use the Gradle wrapper in the source code.

Build Payara Intellij Tools

To build the Payara IntelliJ Tools, run the following command from the root directory where the project was cloned:

If you installed Gradle

1. `gradle build`

If you are using the Gradle wrapper

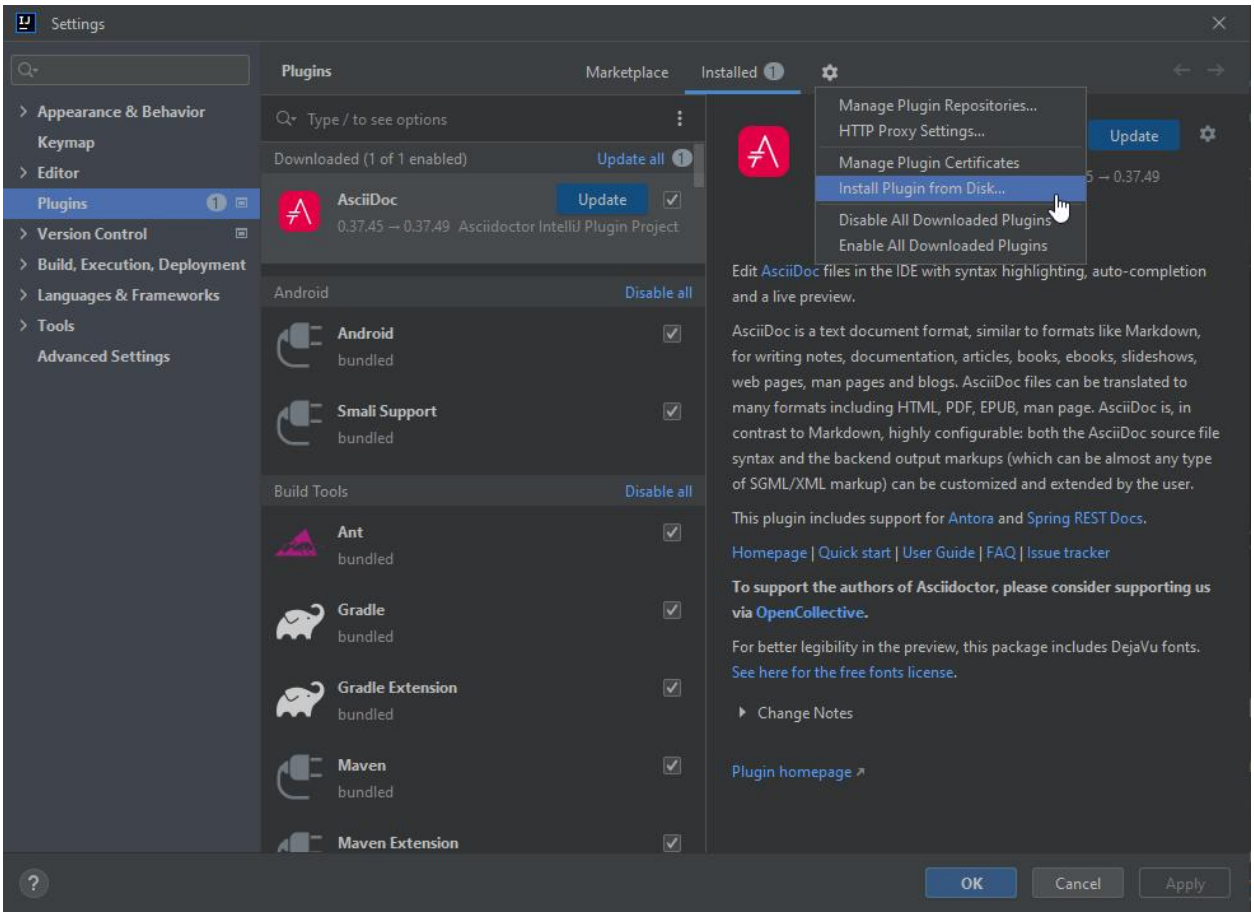
1. `./gradlew build`

After a successful build, the final ZIP distribution will be located in the `/build/distributions` directory. This ZIP file can be installed into IntelliJ IDEA.

Install Payara Tools into Intellij IDEA

After the Payara Tools plugin has been built, you can install it in IntelliJ IDEA as follows:

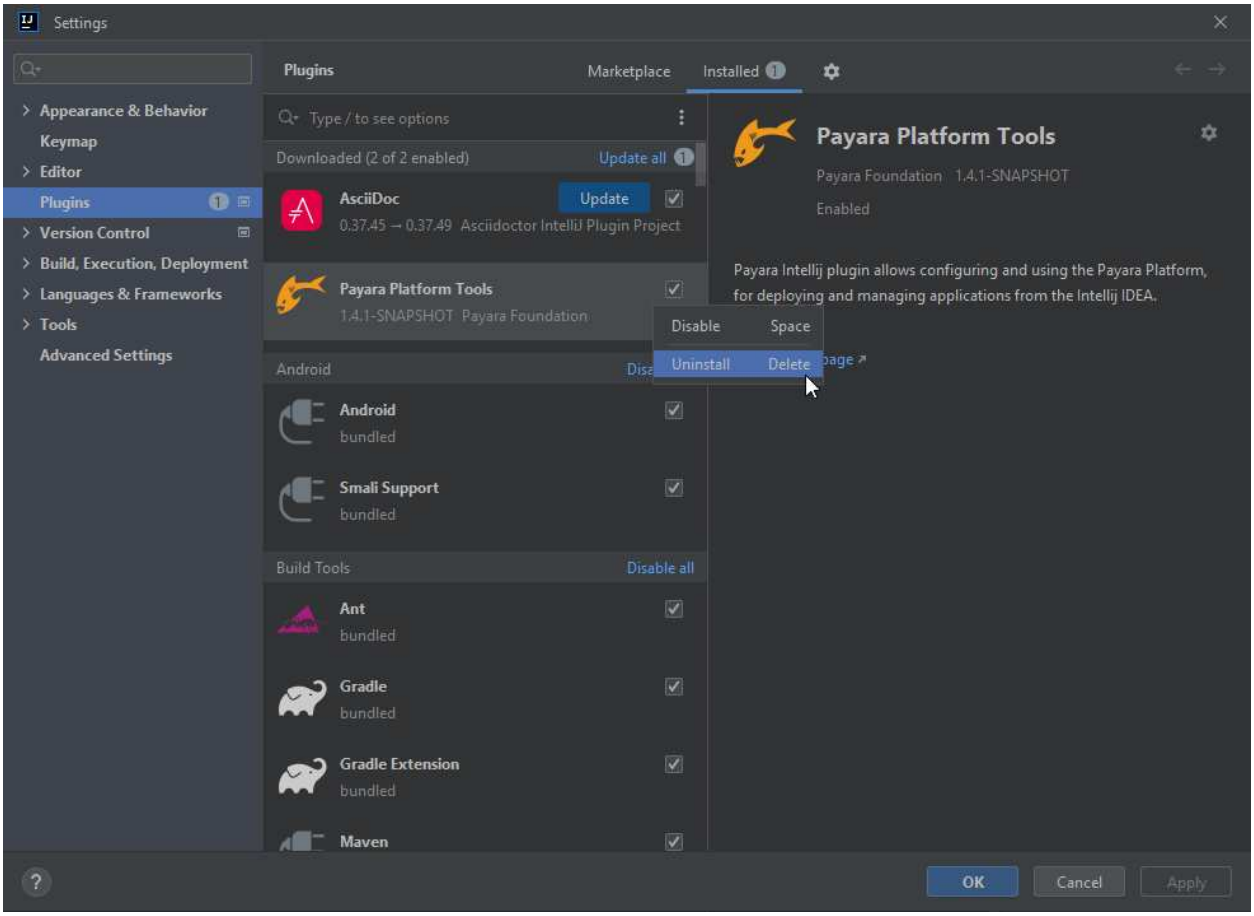
1. Open the **Settings** menu (`Ctrl + Alt + S`)
2. Open the `Plugins` panel
3. Click the ⚙ cog icon and select `Install Plugin from Disk`
4. Select the compiled ZIP file from the `/build/distributions` directory



Removing Payara Tools from Intellij IDEA

To uninstall the plugin:

1. Open the **Settings** menu (`Ctrl + Alt + S`)
2. Open the `Plugins` panel
3. Right-click the **Payara Tools** plugin
4. Click **Uninstall**



Test Payara Tools

Instead of manually installing the plugin each time, you can run a test instance of IntelliJ IDEA with the plugin pre-installed using the following:

If you installed Gradle

1. `gradle runIde`

If you are using the Gradle wrapper

1. `./gradlew runIde`

This will launch a sandboxed IntelliJ IDEA environment with the plugin enabled for testing purposes.

Transform Maven Projects or Files from Java EE 8 to Jakarta EE 10

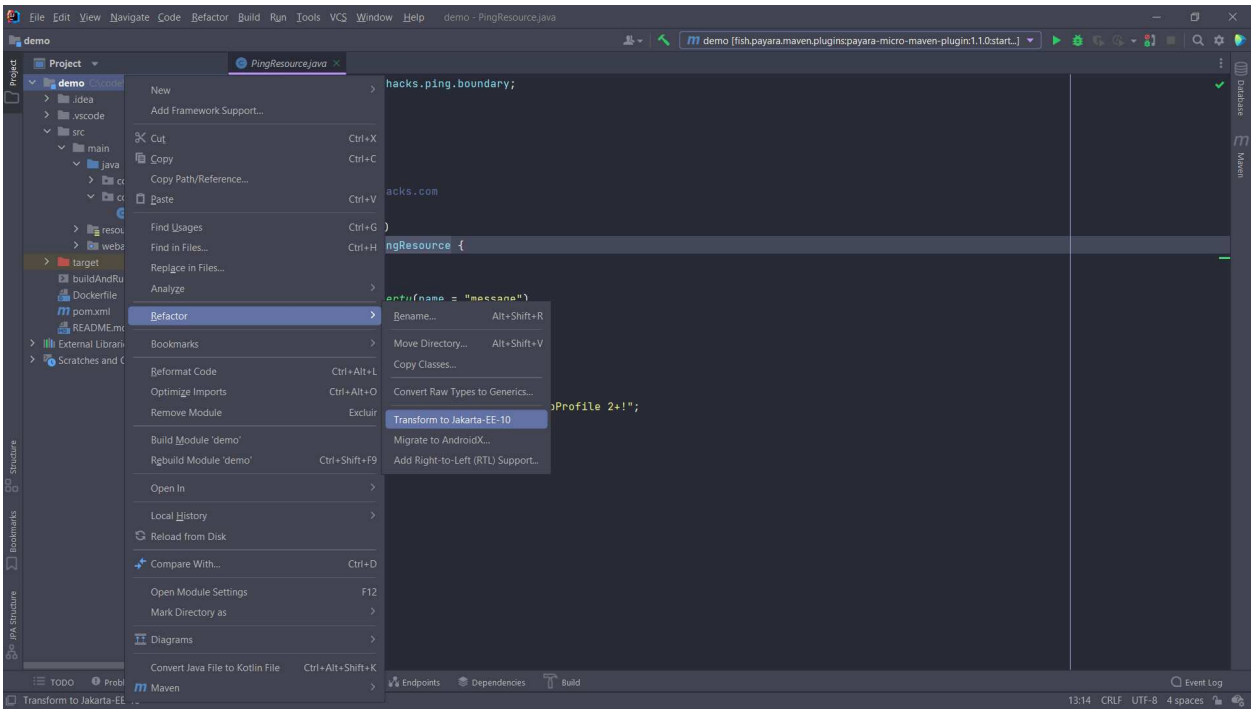
Using the `Payara IntelliJ Ultimate Tools` and `Payara IntelliJ Community Tools` plugins, you can transform whole projects or just a file from Java

EE 8 to Jakarta EE 10.

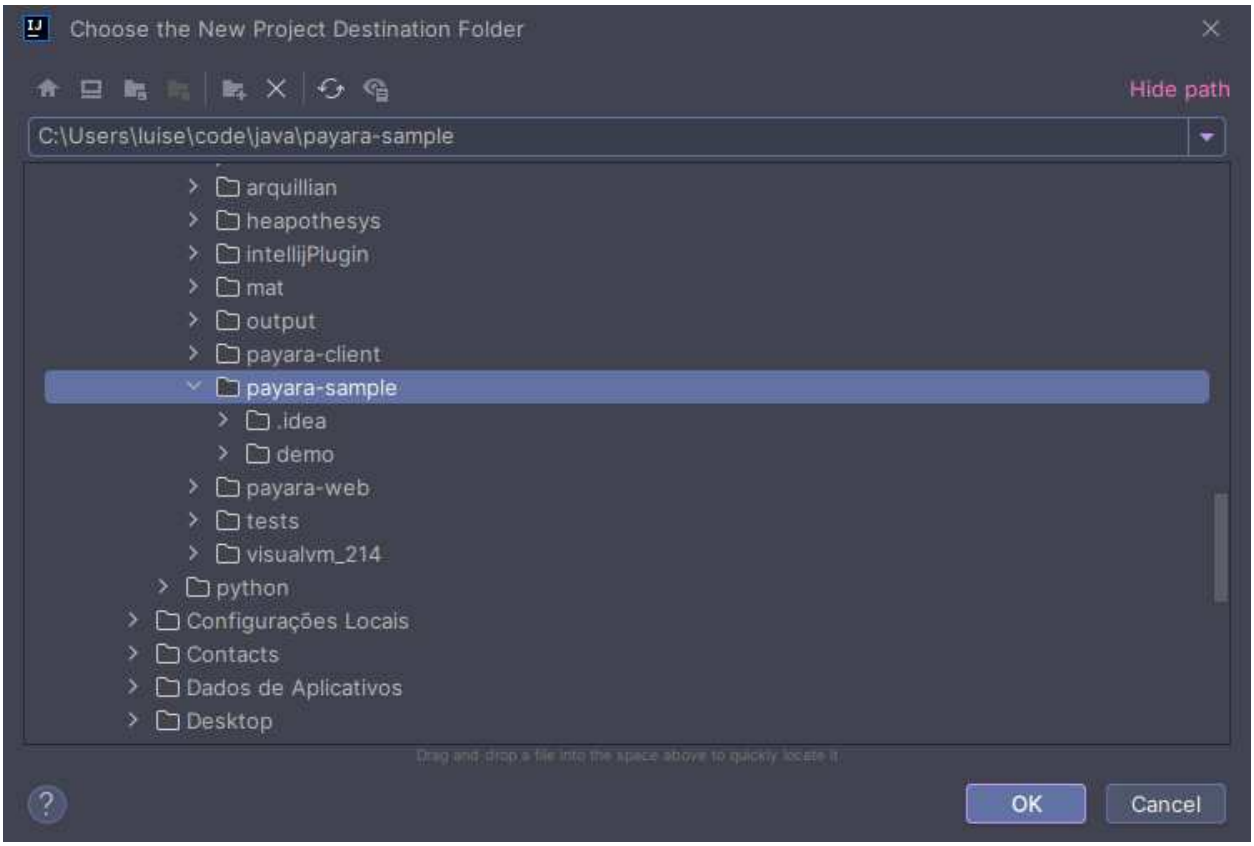
NOTE	This feature is available in both **Community** and **Ultimate** editions of the plugin.

Refactor a Java EE 8 Project to Jakarta EE 10

To refactor a Java EE 8 based **Maven** project right click on root project folder and choose **Refactor** → **Transform to Jakarta EE 10**:



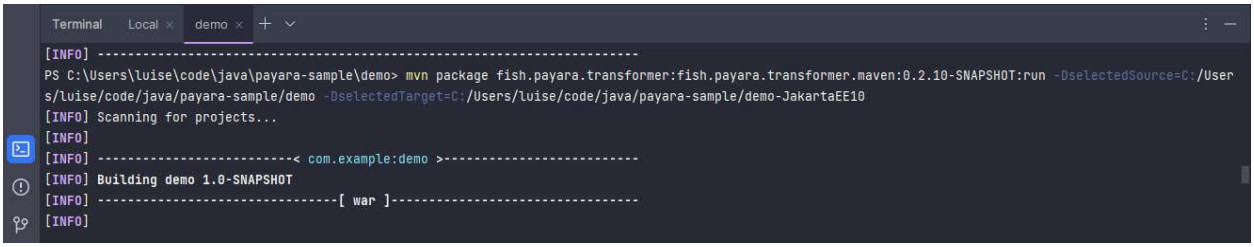
The plugin will open a dialog to choose the destination folder where the new project will be created:



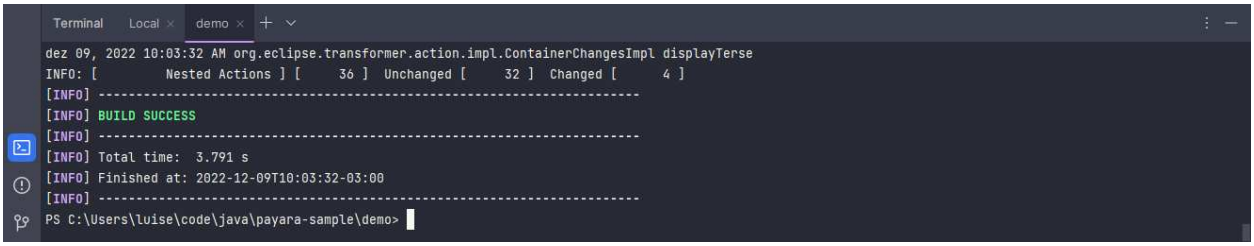
After choosing the `Destination Directory`, a confirmation message will appear indicating that the project is being generated.

> **TIP** | You can see the output of the generation process in the IntelliJ console.

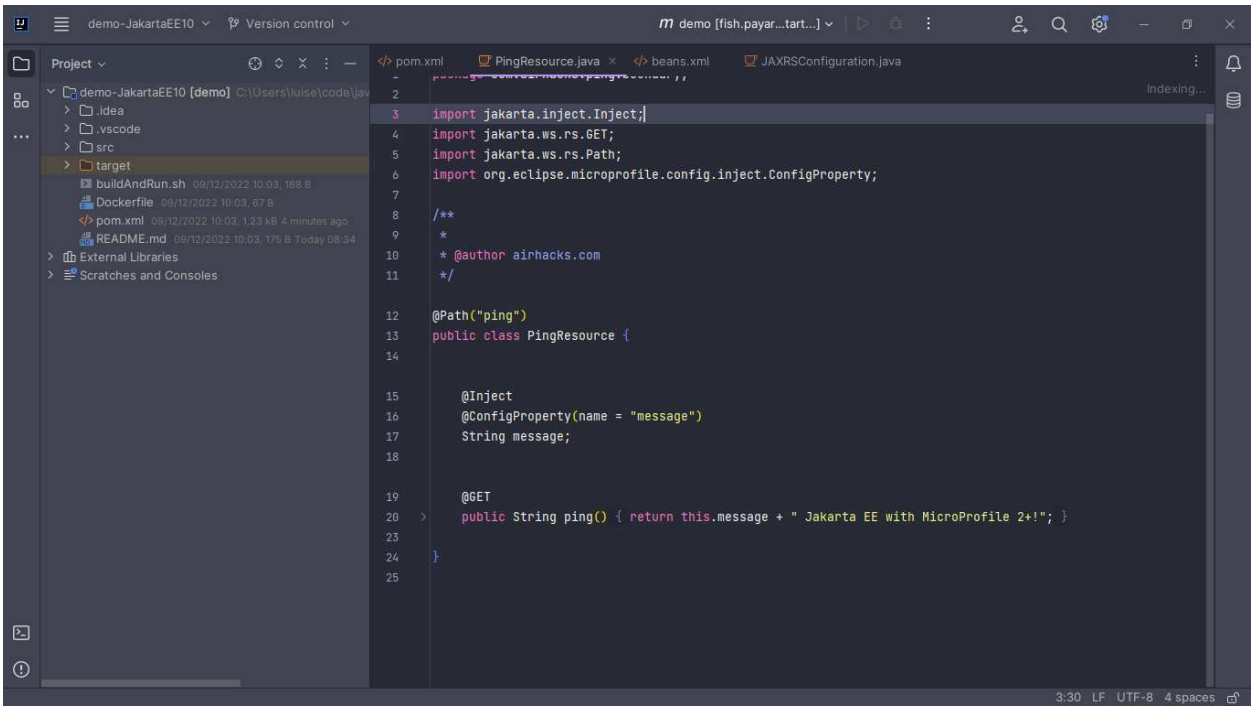
The Maven plugin that executes the transformation process will print its output to the IDE's console:



After the creation, a Maven `BUILD SUCCESS` message should be logged in the IDE's console.



The new project with suffix `-JakartaEE10` is opened in a new IntelliJ instance which loads the project from the Destination Directory previously selected.

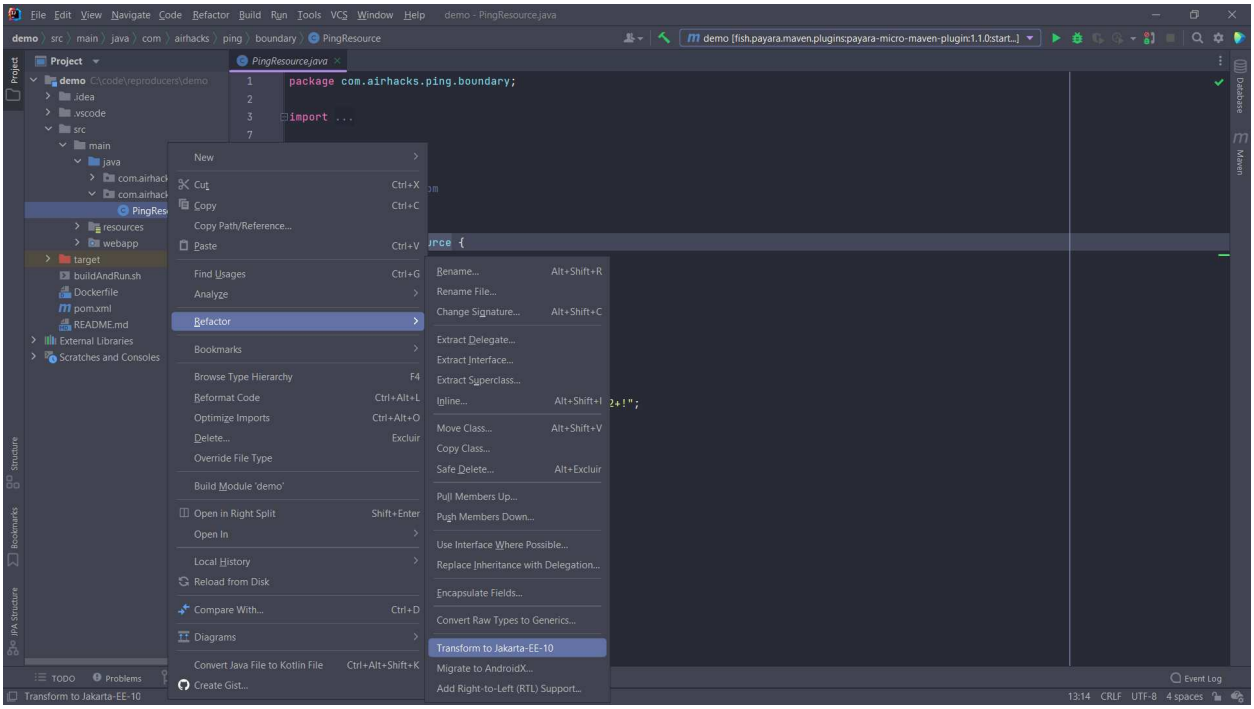


NOTE

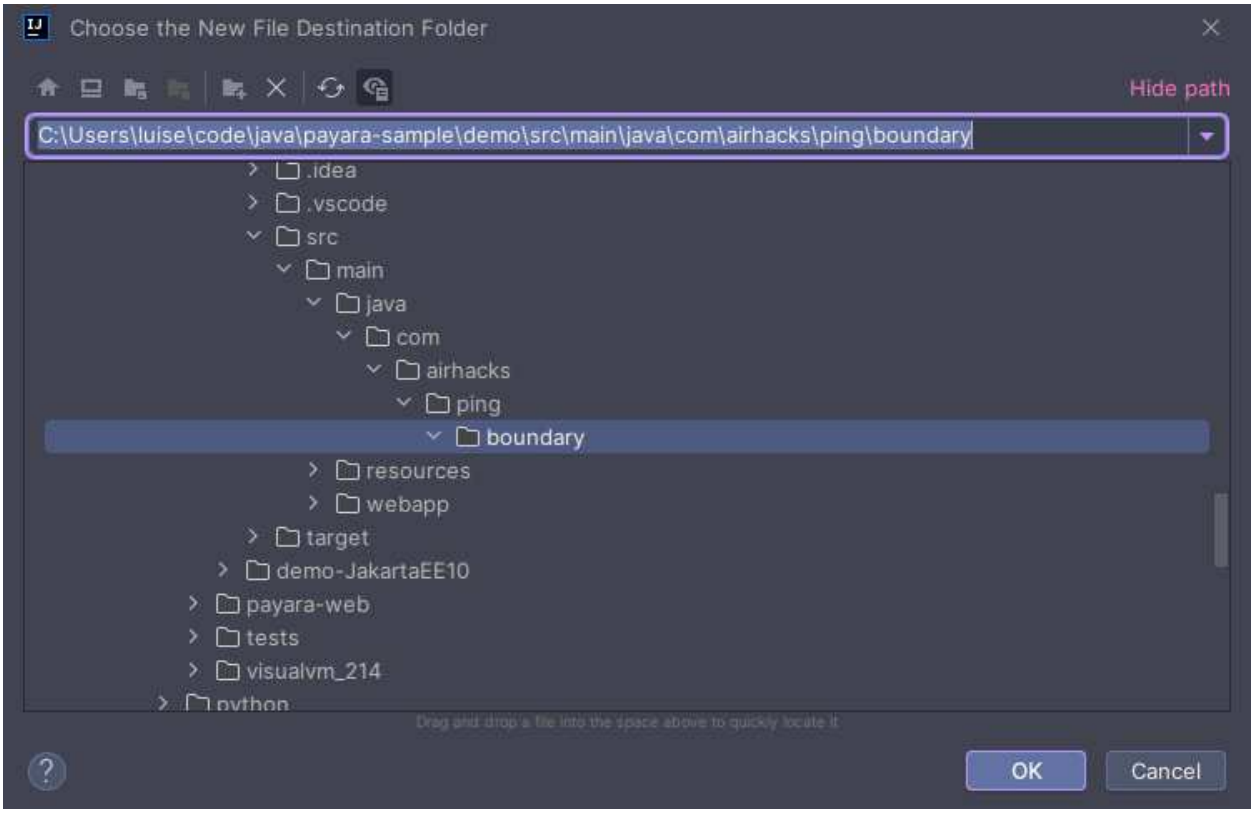
When the whole project is transformed, the plugin adds `-JakartaEE10` at the end of the project name. In our example, `demo-JakartaEE10` project was created from a named `demo`.

Transform a Single File

To refactor a `Java EE 8 file` right click on it and choose **Refactor** → **Transform to Jakarta EE 10** option:



The plugin will open a dialog to choose the destination package where the new file will be created:

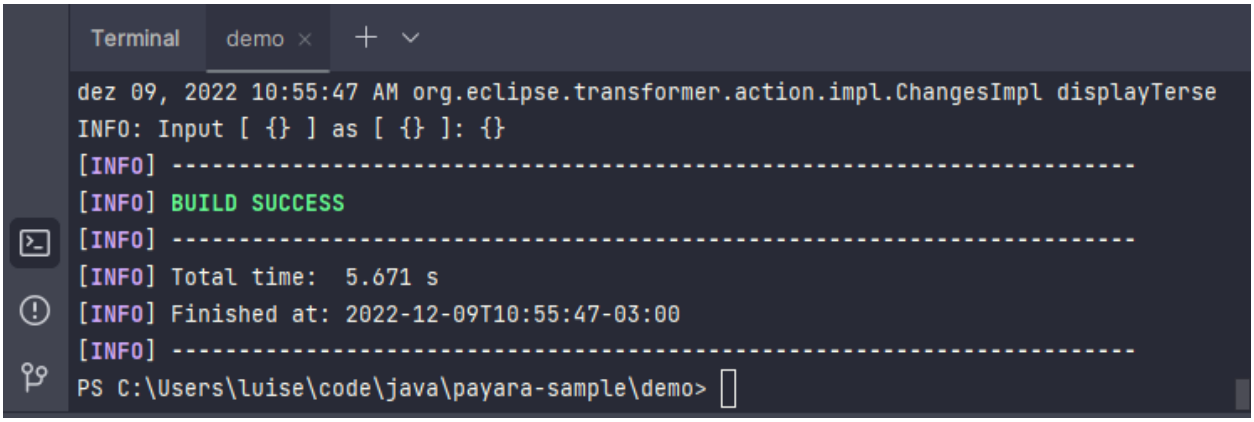


After choosing the `Destination Package`, a confirmation message will appear indicating that the file is being generated.

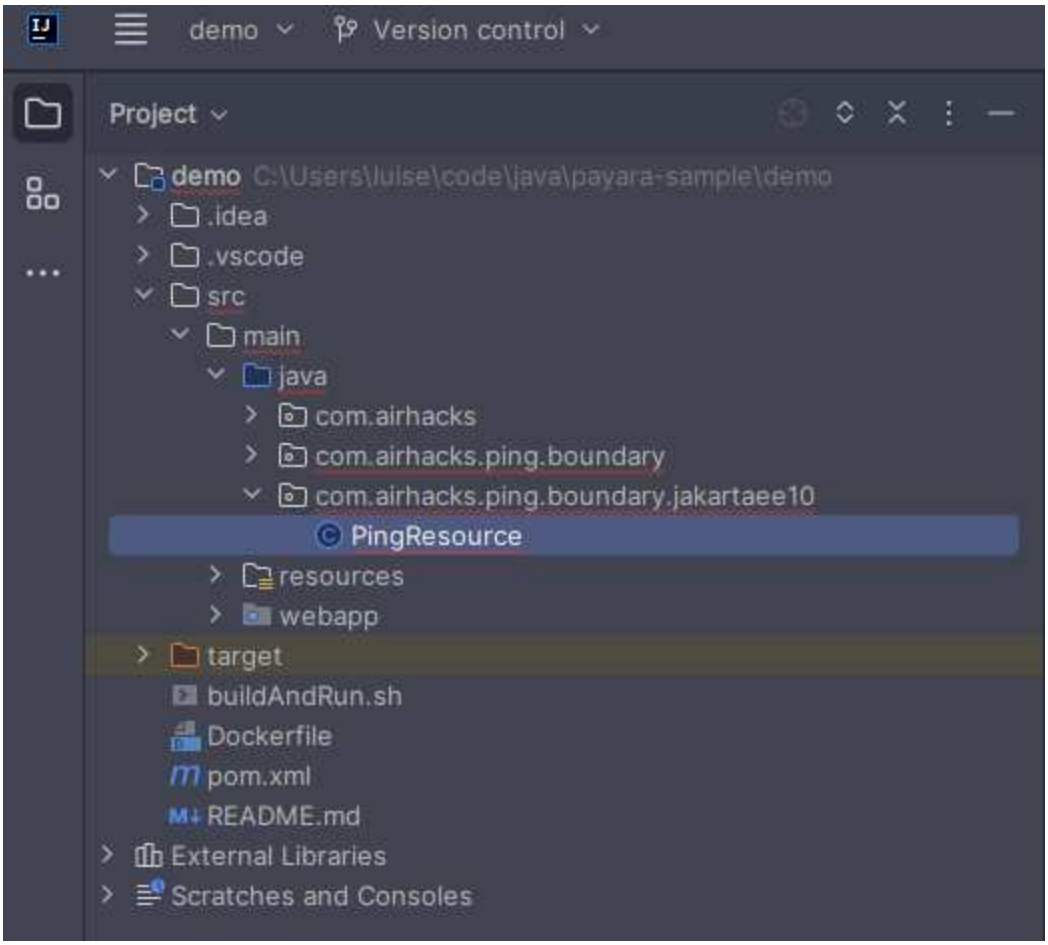
TIP | You can see the output of the generation process in the IntelliJ console.

The output of the Maven plugin that executes the transformation code can be reviewed in the IDE's console:

After the new file is created, a `BUILD SUCCESS` message from the Maven plugin will be logged in the IDE's console.



The new file will be located in the chosen destination package.



NOTE	When a single file is transformed, a new package containing the transformed file will be created in the destination package with the suffix `-jakartaee10`.

Apache NetBeans IDE

Azul Payara Apache Netbeans Tools

Apache NetBeans IDE 16 and higher provides out-of-the-box support for Azul Payara to integrate Payara Server and Payara Micro into the Apache NetBeans IDE's interface.

Therefore, there is no need to manually install the Payara Tools plugin.

Tools

- "Payara Server" - Apache NetBeans enables developers to download the latest version of Payara Server that can be easily configured and added as a recognised server in NetBeans IDE. This means you can easily deploy and manage applications

from the NetBeans IDE.

- "Payara Micro" - Apache NetBeans IDE also provides first-class support to create and run Payara Micro maven web applications from the NetBeans interface. Payara Micro is the microservices-ready version of Payara Server.

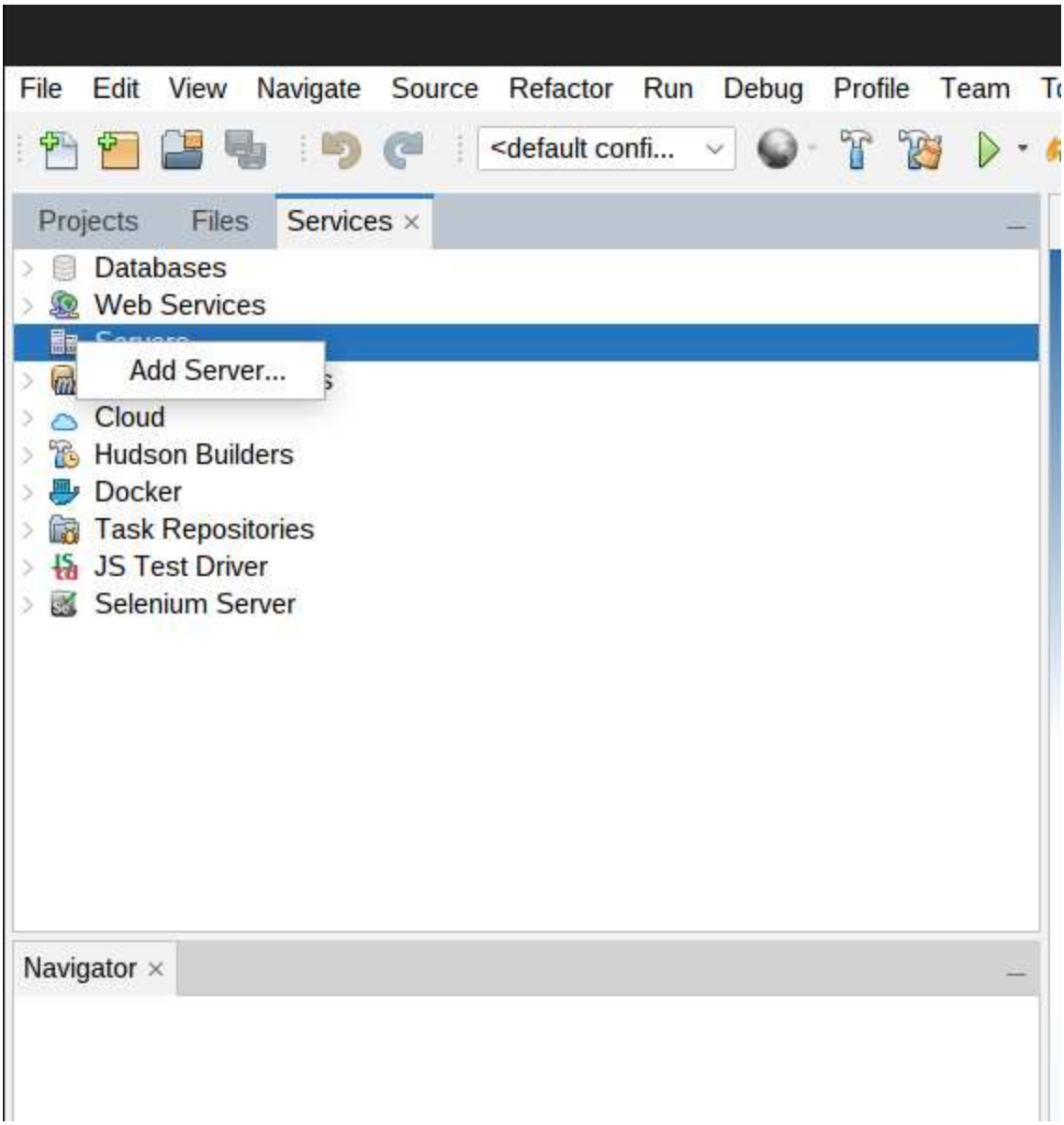
Payara Server Apache Netbeans IDE Support

The Apache NetBeans IDE has built-in integration with Payara Server and allows you to easily perform the following tasks:

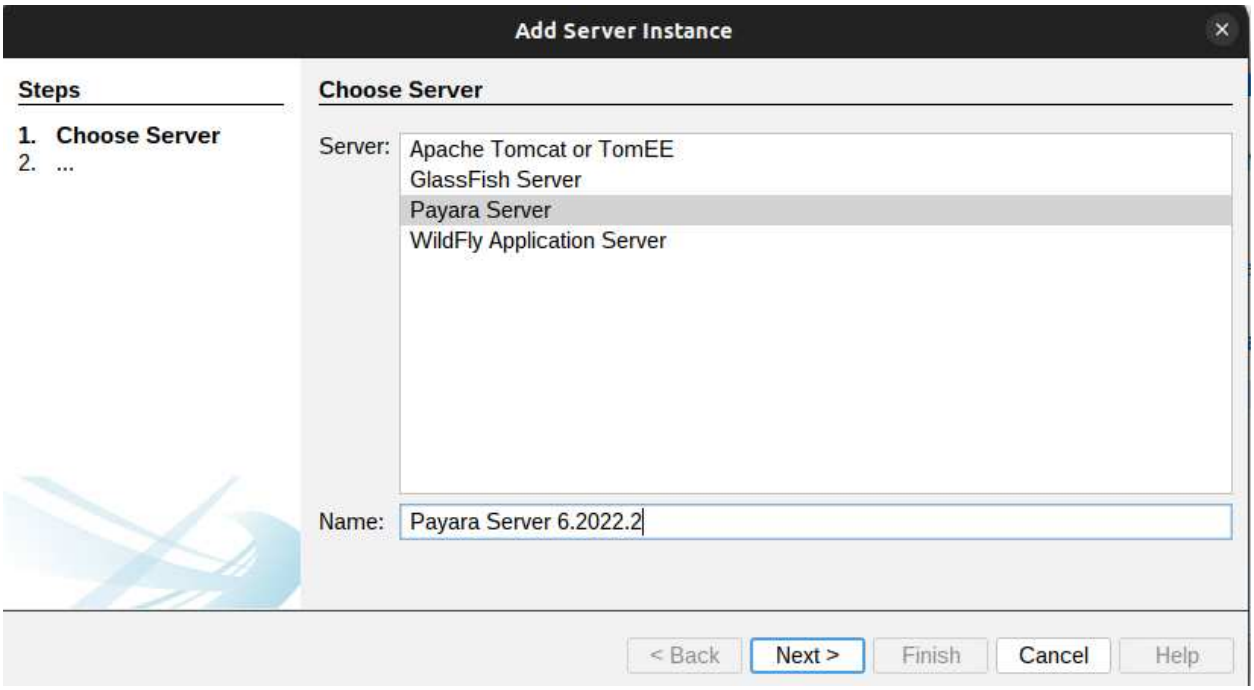
- Start and stop the Payara Server from within the IDE
- Define the Java runtime to be used by Payara Server
- Deploy an application from within the IDE
- Auto Publish & Hot Deploy your application

Adding Payara Server to Apache Netbeans IDE

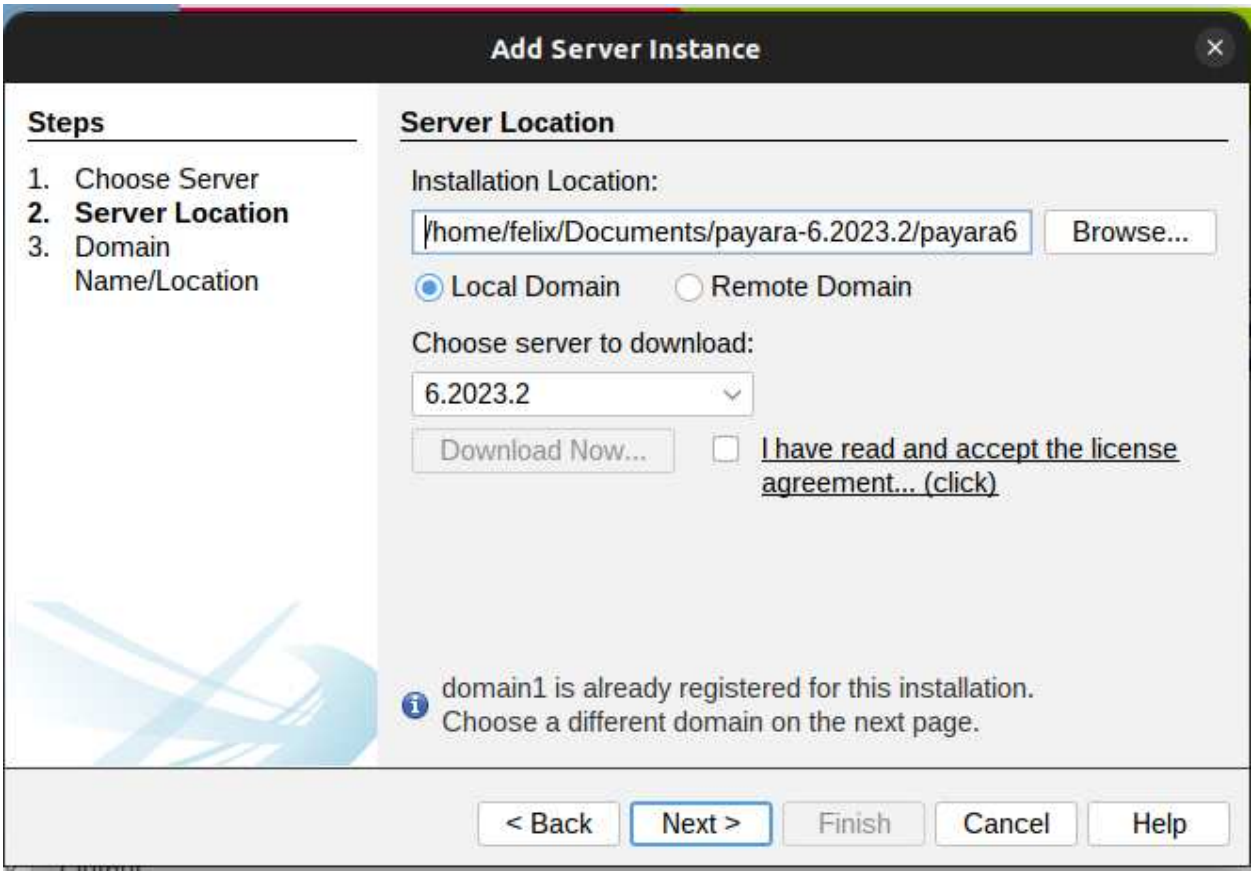
Open the project you want to deploy to Payara Server. Go to the Server tab and right click to add a new server.



Select Payara Server, choose a new name for the server and click Next.

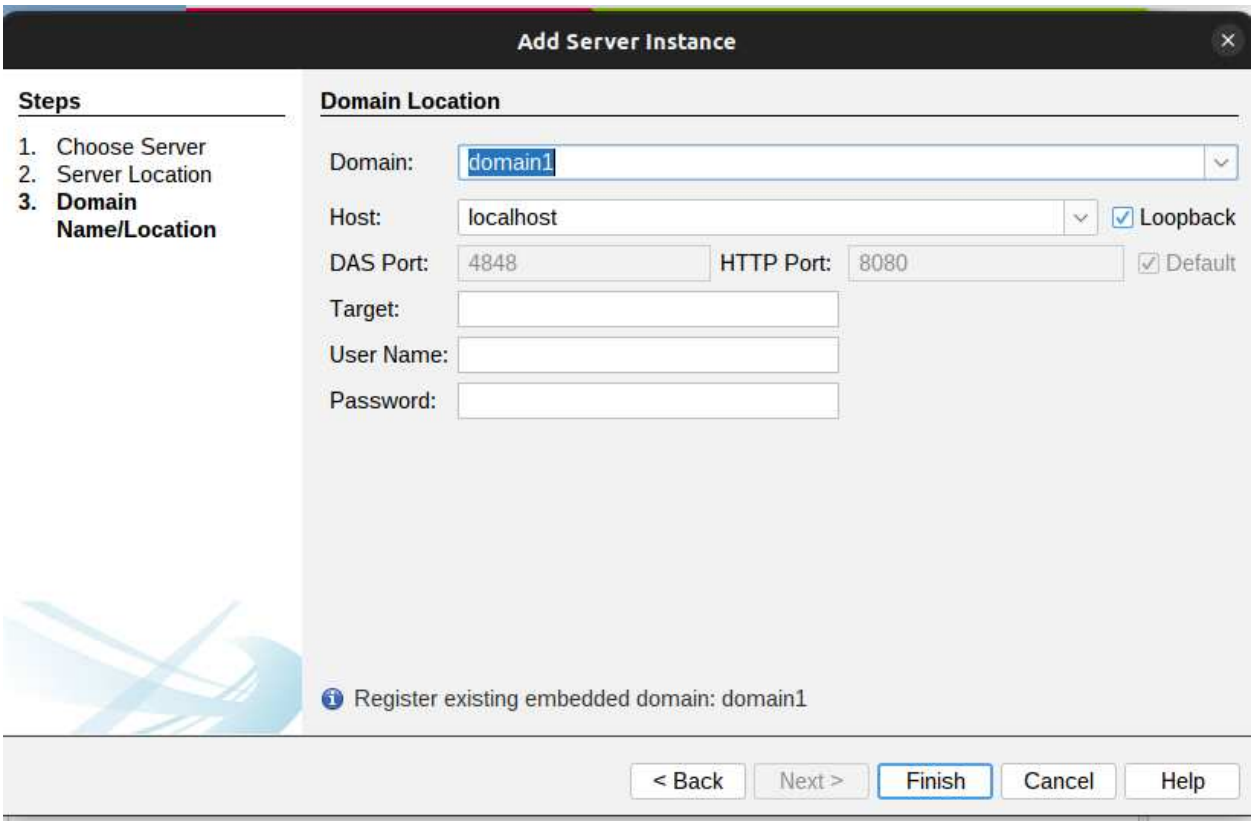


Browse to the location of the Payara Server installation. If the DAS is on another machine, make sure to select the remote domain option. Otherwise, use the local domain option.



The following details are configurable on the next screen:

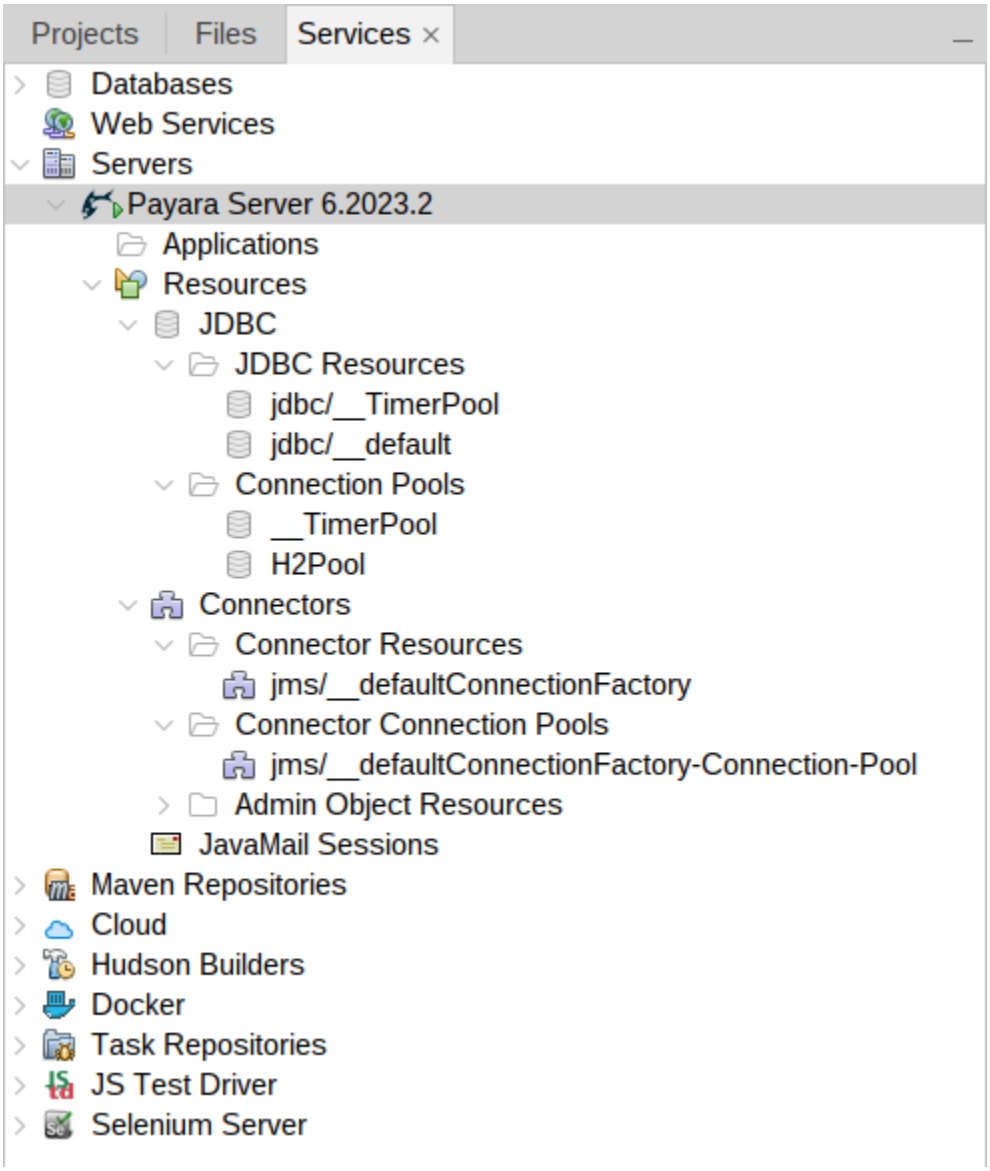
- Domain The domain you want to use. The default domain is **domain1**.
- Host The hostname the server will be listening on. the default hostname is **localhost**.
- DAS Port The port used by the DAS. The default port is **4848**.
- HTTP Port The port used by the deployed applications. The default port is **8080**.
- Target The name of the node that the application will be getting deployed to. This can be left blank for a local installation.
- Username The admin username for Payara Server. The default username is **admin**.
- Password The admin password for Payara Server. The default password is blank.



Managing Payara Server from Apache Netbeans IDE

Once the configuration is complete, you can manage your Payara Server directly from Apache NetBeans. By right-clicking on the previously-configured server, we can do the following actions:

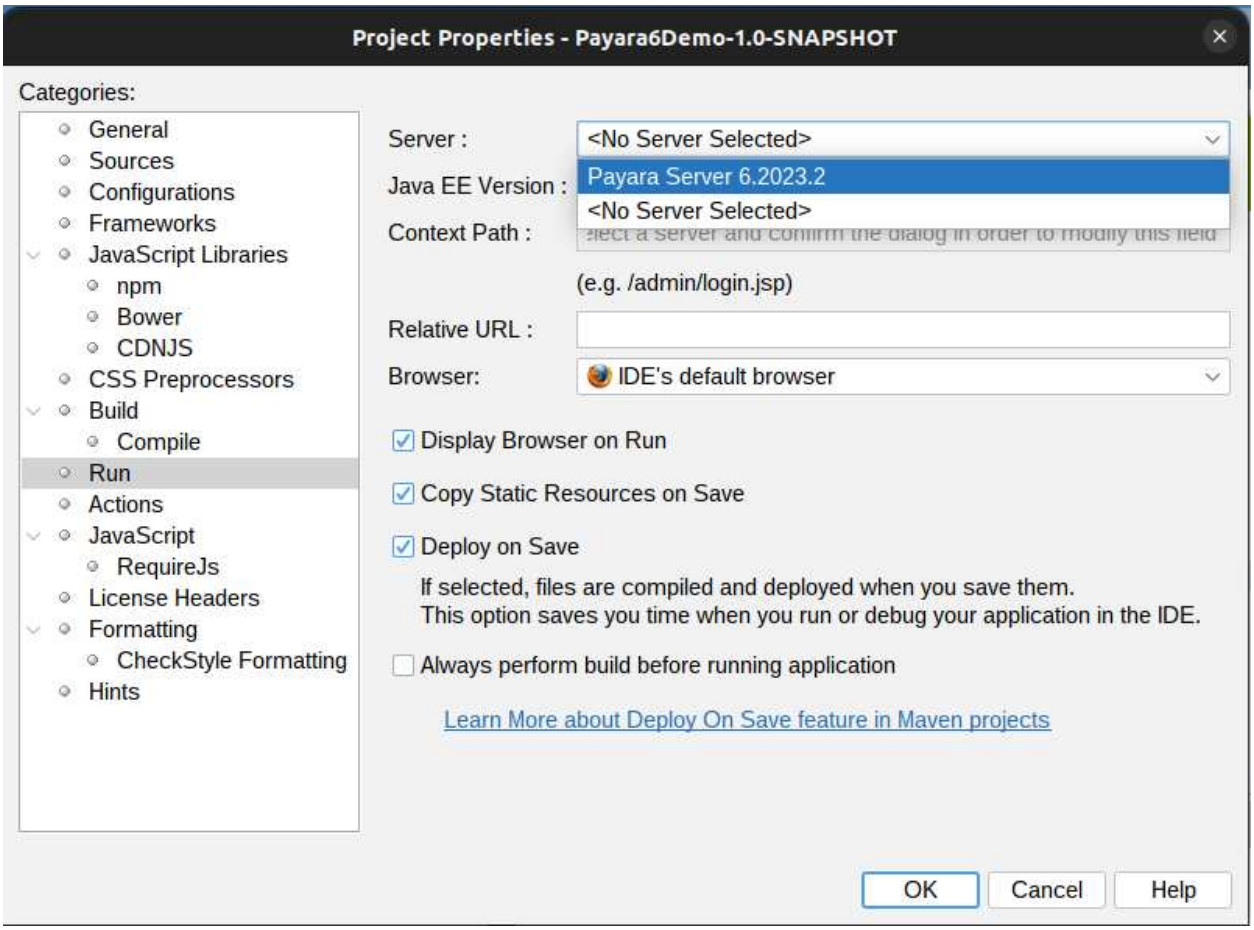
- Start or stop the server
- Start debugging the server
- Open the admin console
- Open the server.log file



Exploding the server dropdown will list Applications, Resources and Web Services. This allows you to view and manage certain Payara Server configurations, from the NetBeans interface.

Deploying Applications to Payara Server from Apache Netbeans IDE

An application can be deployed and un-deployed easily from within Apache NetBeans. First, the application run settings need to be configured to use Payara Server. Go to Properties on a project and then the Run section of this window. This menu lists the configuration for deploying the application.



Select the Payara Server you have configured. You can also select a context path for the application to be deployed to, as well as NetBeans behaviour when deploying.

Deploying to Payara Server now only requires running the application like any other NetBeans project. Once it's deployed, the Services view will then display the application under Payara Server, where you can undeploy, enable, disable and view the application in the browser.

Deploy Applications in a Local Docker Container

The Payara Server tools for Apache NetBeans include Docker integration, making it

seamless to deploy exploded applications to a Payara Server instance running within a local Docker container.

To take advantage of this feature, follow these steps below:

1. Open Apache NetBeans and navigate to the *Services* tab.
2. Register a new Payara Server instance by performing the following:
   a. Click on the "*Add Server*" button.
   b. Select the "*Remote Domain*" option.
   c. Browse and locate the Payara Server installation directory with the same version as the one running in the Docker container.
   d. Enter the domain name, host, and necessary credentials.
   e. For the Docker integration, choose the "*Docker Volume*" option and provide the `HOST_PATH` (e.g., `D:/test`) and `CONTAINER_PATH` (e.g., `/mnt/test`) mapping details.

      This mapping should align with the exploded application path or parent path of exploded application.
3. Run the Payara Server Docker container using the following command:

   ```
   docker run -p 8080:8080 -p 4848:4848 -p 9009:9009 -v HOST_PATH:CONTAINER_PATH
   payara/server-full:{release-id}
   ```

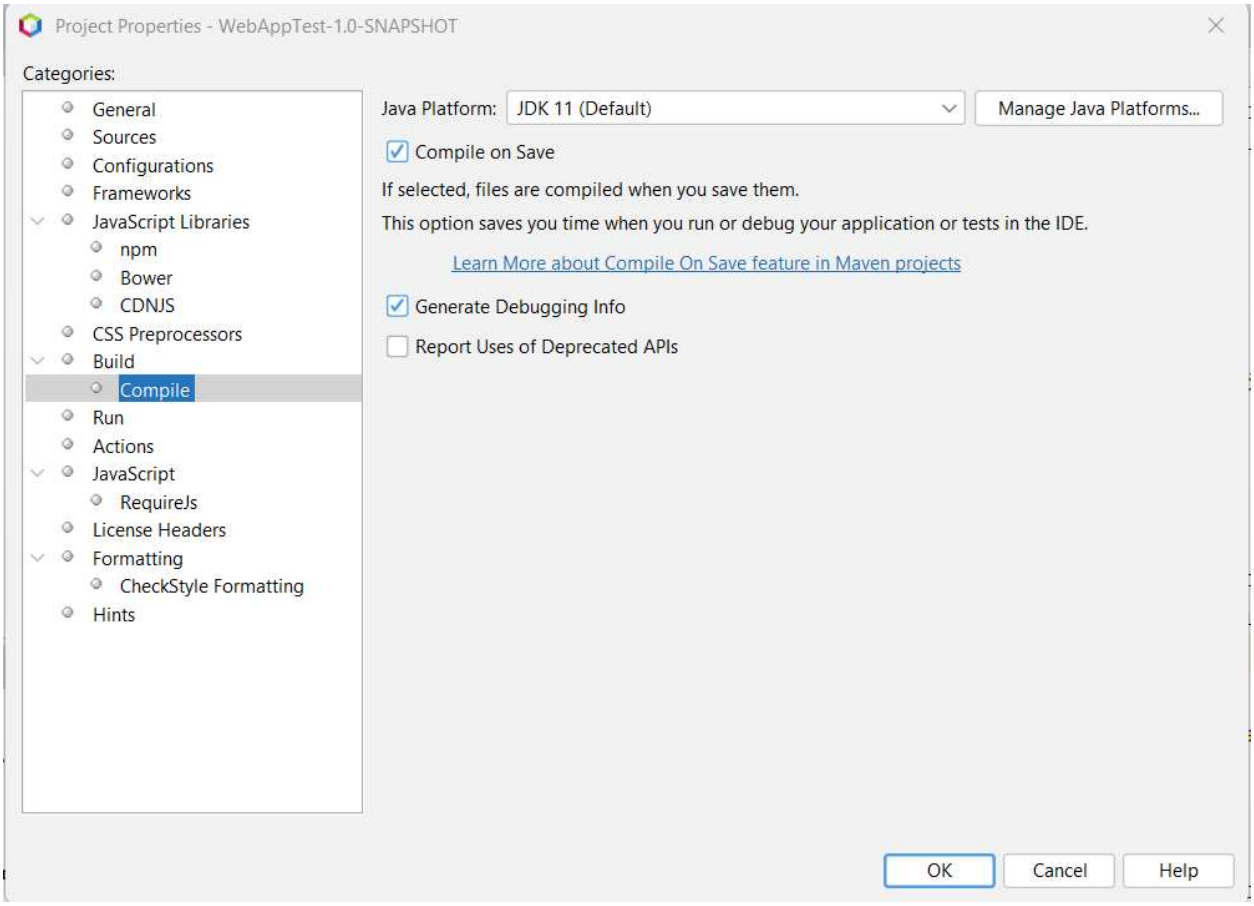
   Replace `HOST_PATH` with the actual host path and `CONTAINER_PATH` with the desired container path.

   The `HOST_PATH` points to the location on your host machine, and `CONTAINER_PATH` is the corresponding location inside the container.
4. Create your web application and deploy it to the registered Payara Server instance.

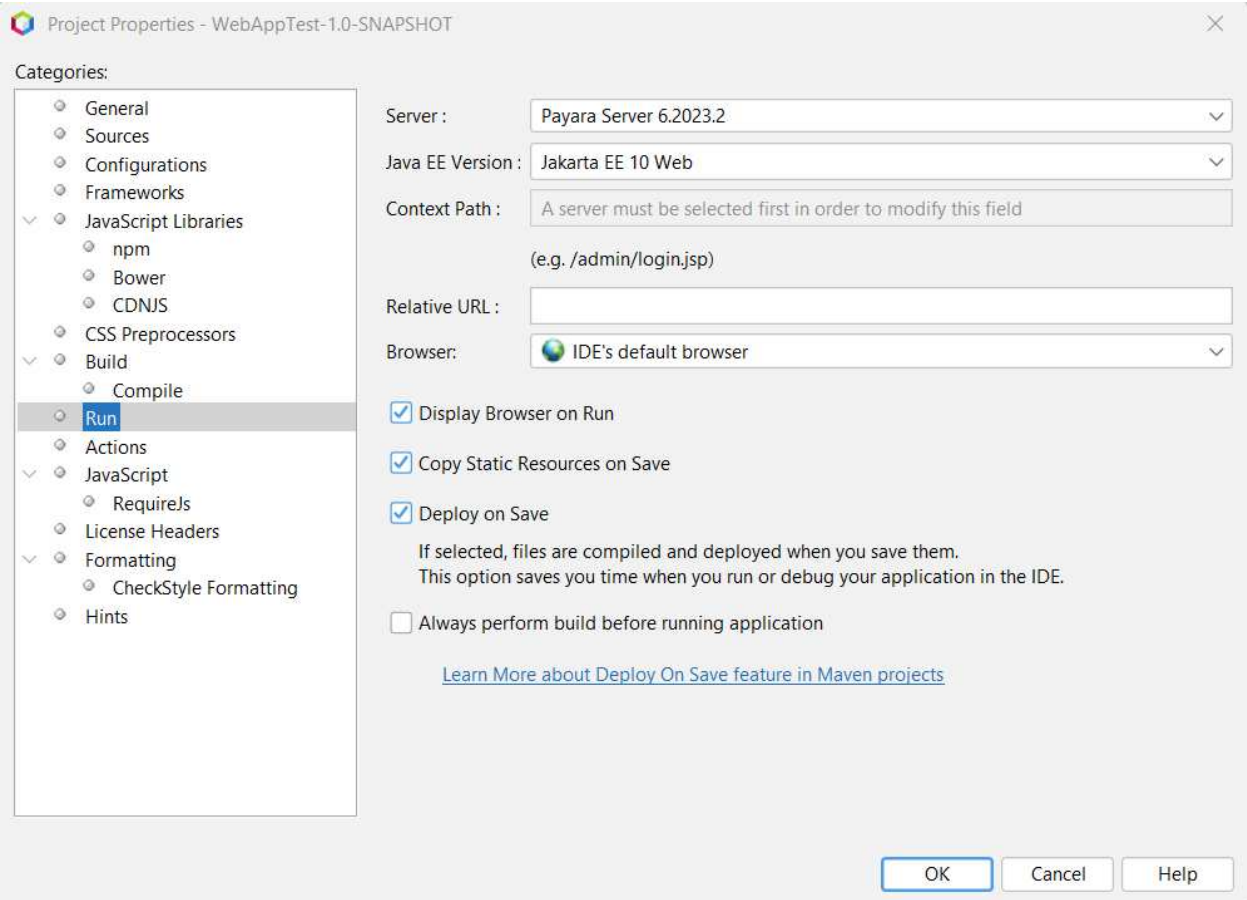
Hot Deploy and Auto Deploy Settings

To enable the **Auto Deploy** features for Payara Server on the Apache NetBeans IDE, you must enable the **Compile on Save** and **Deploy on Save** features from the **Project Properties**.

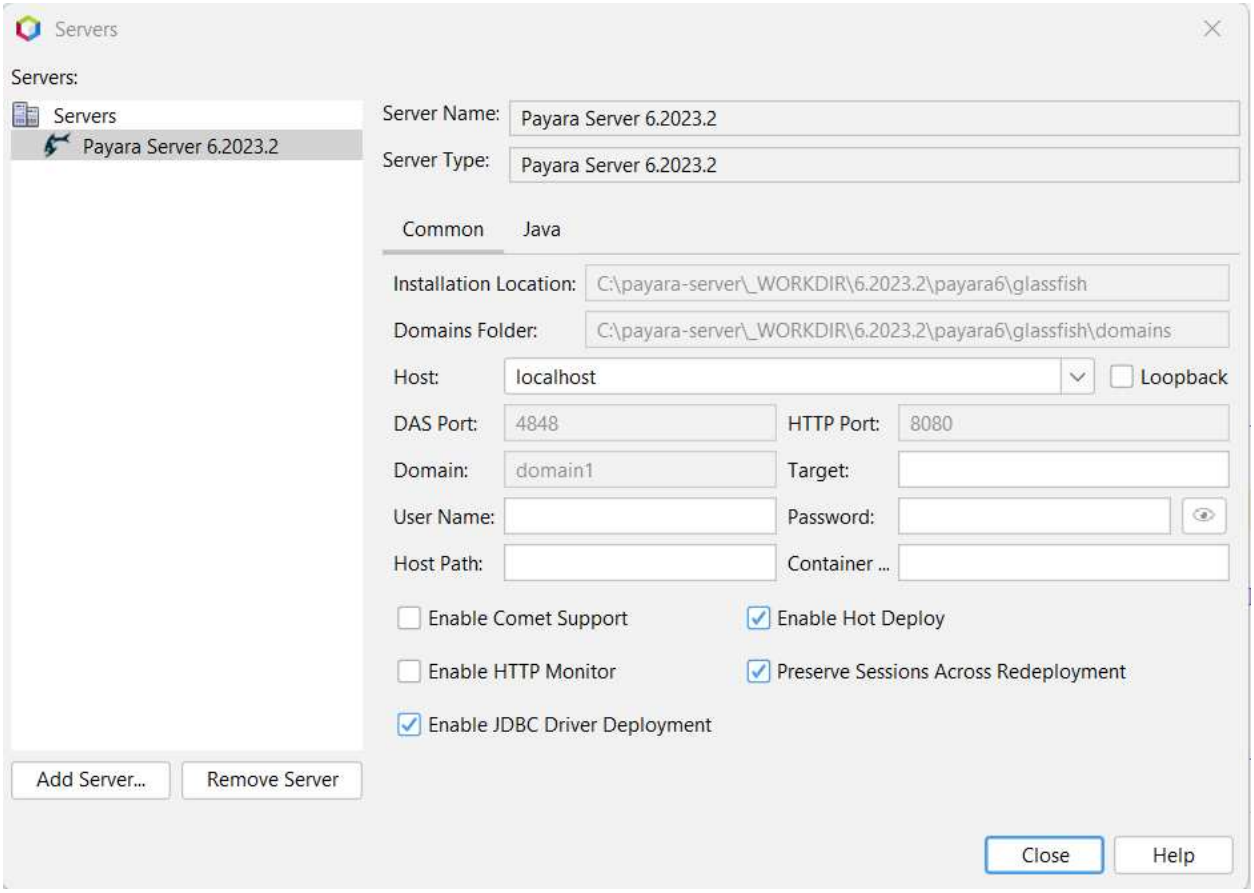
Compile on Save can be found in the **Build**▯**Compile** submenu of the project properties.



Deploy on Save can be found in the Run submenu of the project properties.



To enable the **Hot Deploy** feature for Payara Server on the NetBeans IDE, you must right-click the Server under the Services tab and select **Properties**. From the Commons tab check the Enable Hot Deploy checkbox.



See Also

"Hot Deploy and Auto Deploy"

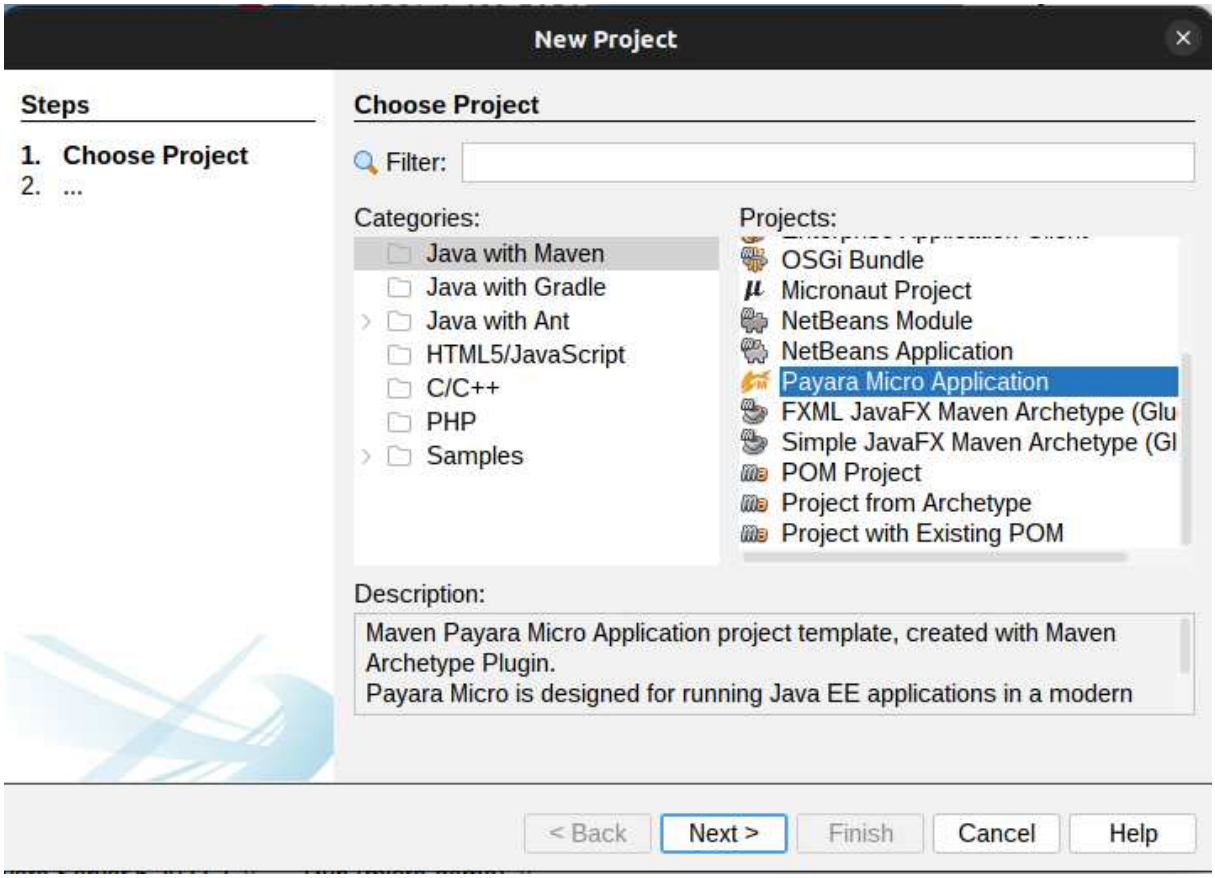
Payara Micro Apache Netbeans IDE Support

Apache NetBeans has Payara Micro support built into the IDE, so there is no need to install a plugin to integrate and control Payara Micro from Apache NetBeans.

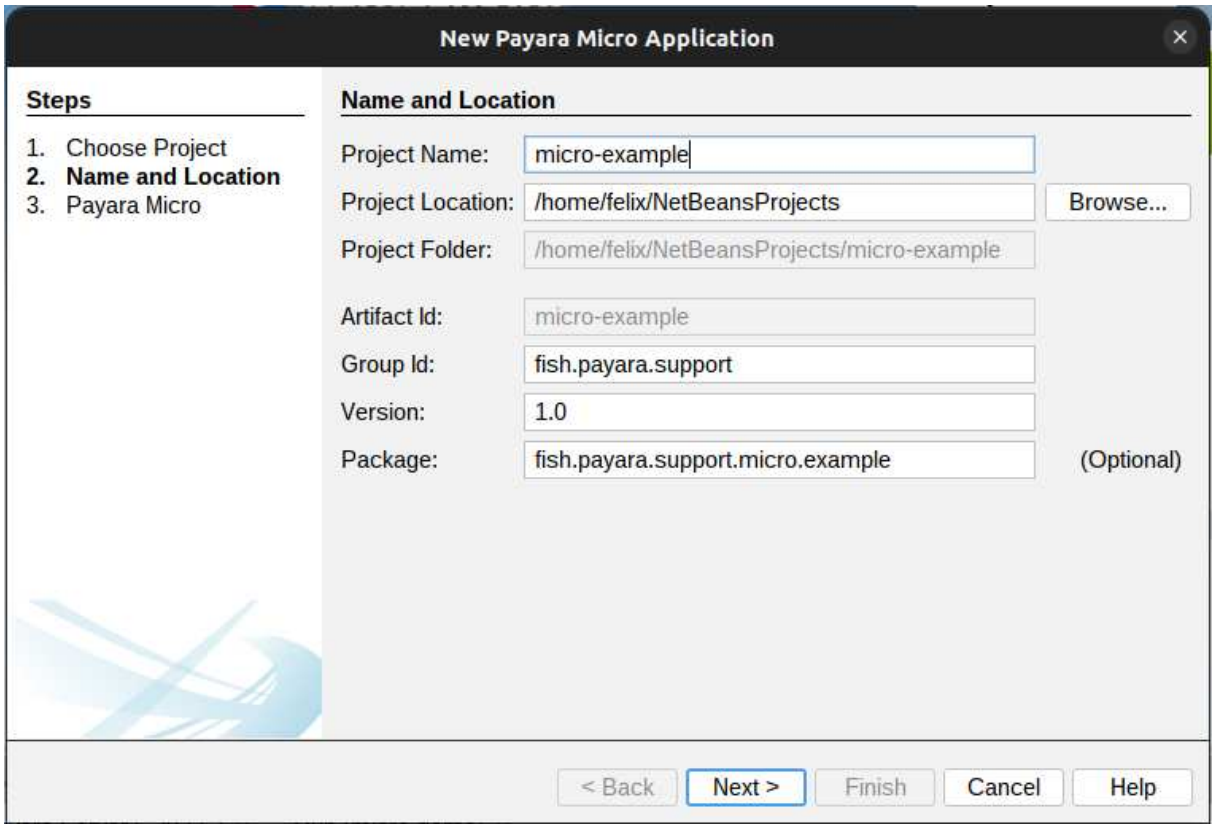
Creating a Payara Micro Project

Apache NetBeans IDE supports Payara Micro out of the box. To create a Payara Micro project, follow these steps:

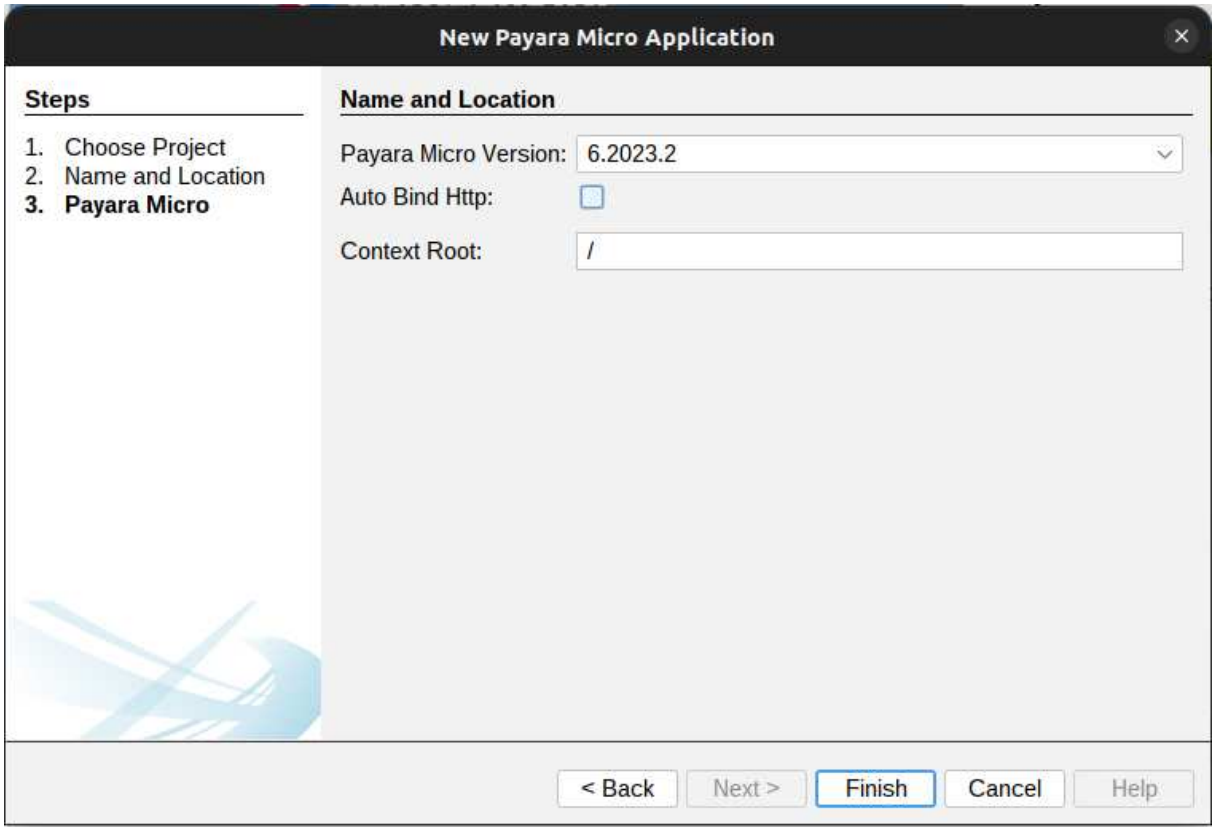
1. Create a new project using Payara Micro Application archetype under Java with Maven category.



2. Input the desired project name, project name, artifact id, group id and version.



3. Select the desired Payara Micro version. Make sure that Auto Bind HTTP is left unchecked.



Deploying the Payara Micro Project

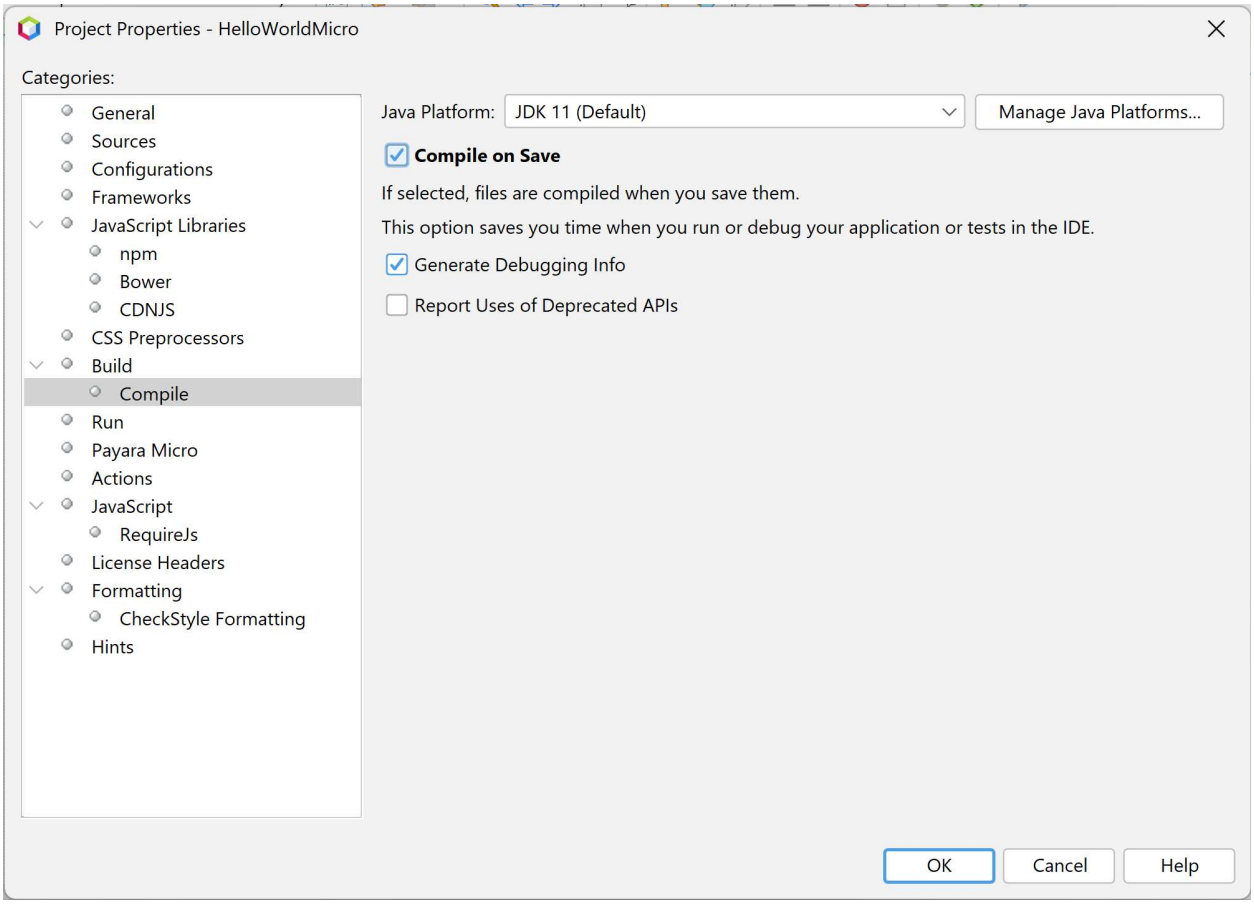
To deploy the Payara Micro project, simply run the application using the Run Project option from the Run menu. The Payara Micro application will be deployed and the URL will be displayed in the *Output* window:

```
[2023-02-28T19:42:03.889+0000] [] [INFO] [AS-WEB-GLUE-00198] [jakarta.enterprise.web] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1677613323889] [levelValue: 800] Created HTTP l

[2023-02-28T19:42:03.905+0000] [] [INFO] [AS-WEB-GLUE-00200] [jakarta.enterprise.web] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1677613323905] [levelValue: 800] Created virtua

[2023-02-28T19:42:04.071+0000] [] [INFO] [AS-WEB-GLUE-00201] [jakarta.enterprise.web] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1677613324071] [levelValue: 800] Virtual server

[2023-02-28T19:42:04.761+0000] [] [INFO] [] [javax.enterprise.system.core] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1677613324761] [levelValue: 800] MicroTest2-1.0-SNAPSHOT w

[2023-02-28T19:42:04.762+0000] [] [INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1677613324762] [levelValue: 800] Deployed 1 archive(s)

[2023-02-28T19:42:04.804+0000] [] [INFO] [AS-WEB-GLUE-00130] [jakarta.enterprise.web] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1677613324804] [levelValue: 800] Invalid Sessio

[2023-02-28T19:42:04.823+0000] [] [INFO] [AS-WEB-GLUE-00172] [jakarta.enterprise.web] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1677613324823] [levelValue: 800] Loading applic

[2023-02-28T19:42:04.835+0000] [] [INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1677613324835] [levelValue: 800] {
    "Instance Configuration": {
        "Host": "host.docker.internal",
        "Http Port(s)": "8080",
        "Https Port(s)": "",
        "Instance Name": "Laconic-Wahoo",
        "Instance Group": "MicroShoal",
        "Hazelcast Member UUID": "0ea97503-f5ac-4529-8d93-d06c15e90d7e",
        "Deployed": [
            {
                "Name": "MicroTest2-1.0-SNAPSHOT",
                "Type": "war",
                "Context Root": "/MicroTest2-1.0-SNAPSHOT"
            }
        ]
    }
}

[2023-02-28T19:42:04.837+0000] [] [INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1677613324837] [levelValue: 800]
Payara Micro URLs:
http://host.docker.internal:8080/MicroTest2-1.0-SNAPSHOT


[2023-02-28T19:42:04.837+0000] [] [INFO] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1677613324837] [levelValue: 800] Payara Micro 6.2023.2 #badassmicrofish (bu
```

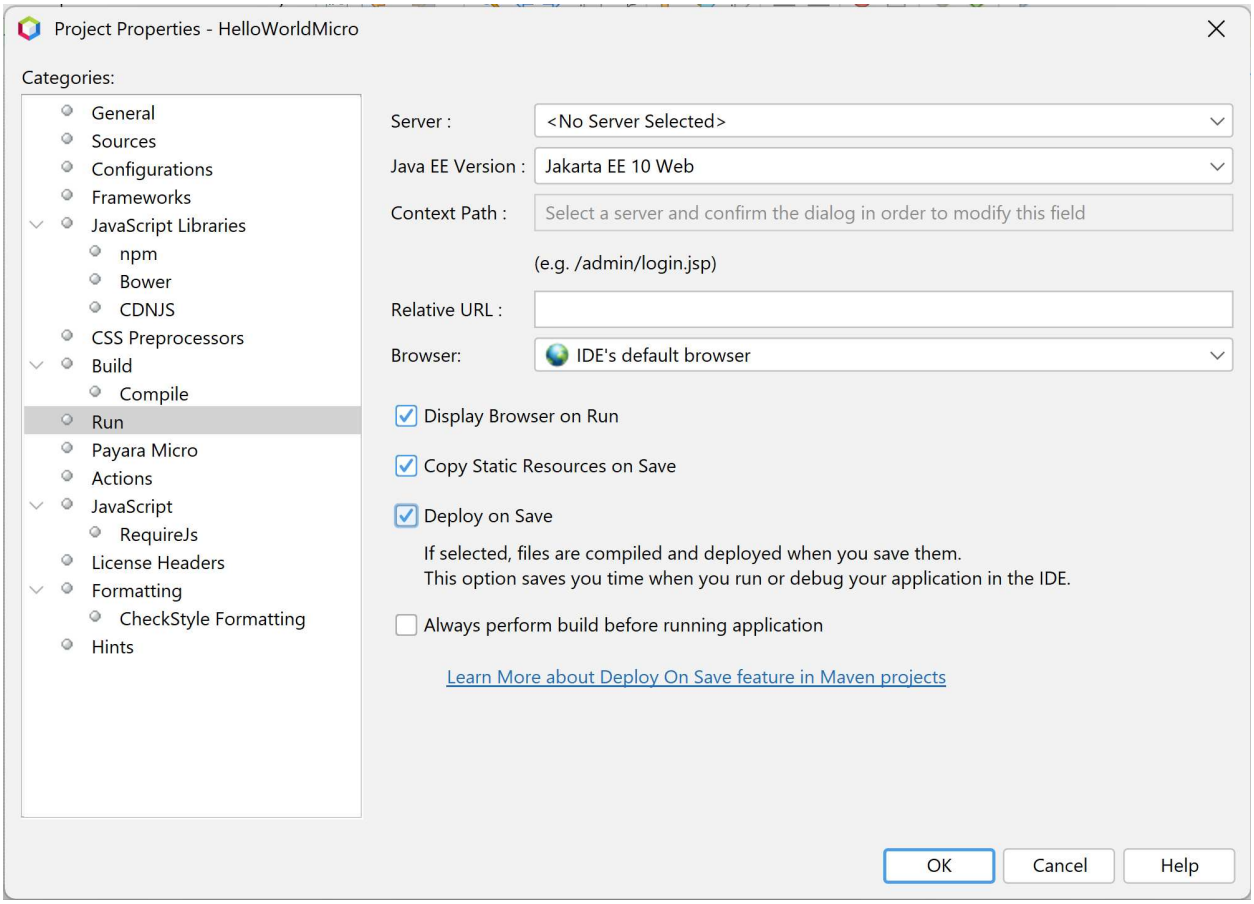
Hot Deploy and Auto Deploy Settings

To enable the **Auto Deploy** features for Payara Micro on the Apache NetBeans IDE, you must enable the **Compile on Save** and **Deploy on Save** features from the **Project Properties**.

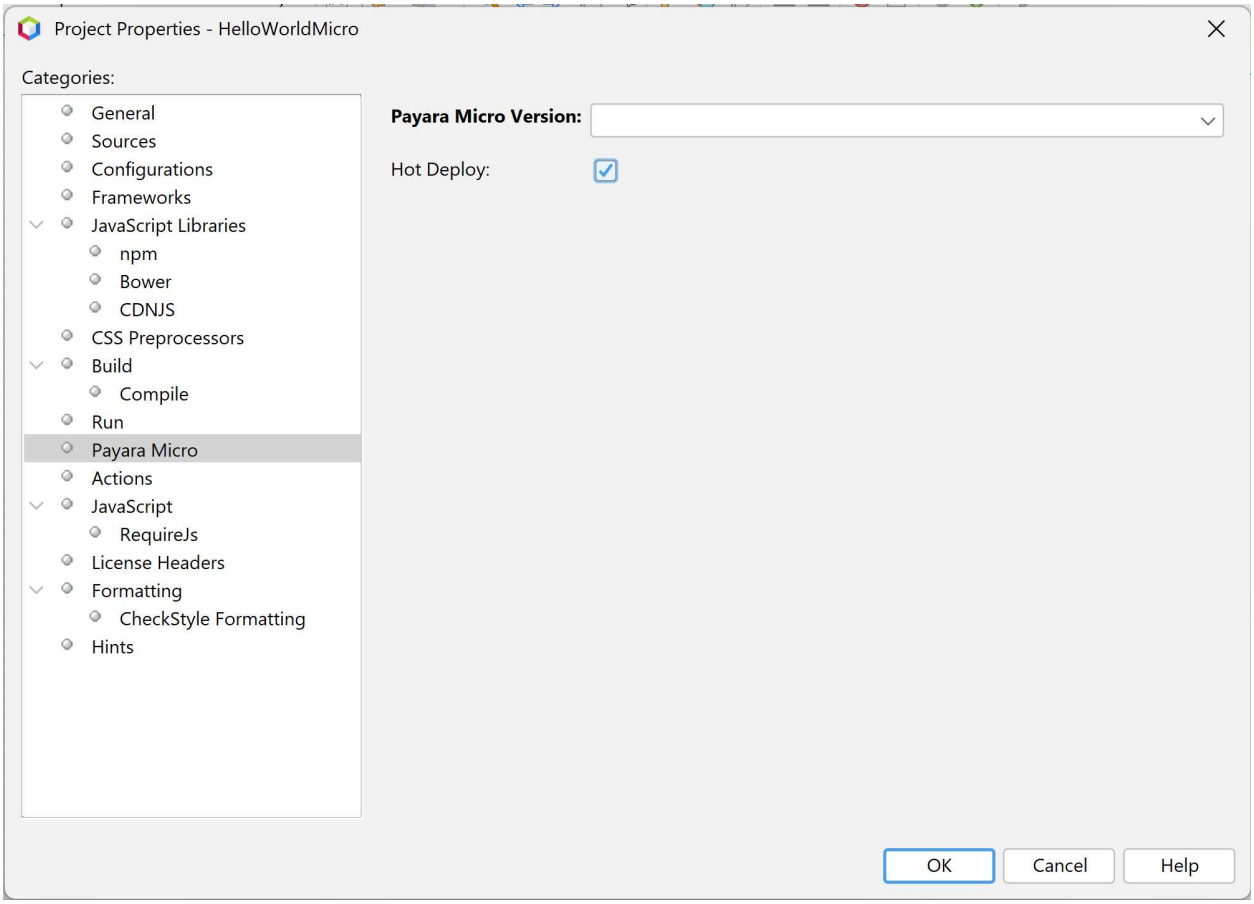
Compile on Save can be found in the **Build**→**Compile** submenu of the project properties.



Deploy on Save can be found in the Run submenu of the project properties.



To enable the **Hot Deploy** feature for Payara Micro on the Apache NetBeans IDE, you must enable the **Hot Deploy** checkbox which can be found in the **Payara Micro** submenu of the project properties.



See Also

"Hot Deploy and Auto Deploy"

Building Payara Tools Netbeans IDE Plugin

As Apache NetBeans IDE provides out-of-the-box support for Azul Payara, the source code for the Azul Payara tools is contained within the NetBeans Code Repository.

Full, up-to-date build instructions for the Apache NetBeans IDE can be read on the README file in the Apache NetBeans IDE repository

Prerequisites

Before building the Apache NetBeans IDE and subsequently the Azul Payara Tools for Apache NetBeans IDE, you will need to configure your local environment. Subsequent builds will not require these steps.

- **Clone the Apache NetBeans Repository**: You must first clone the repository where the plugin source code is stored. Apache NetBeans Code Repository

- **Install Apache Ant**: Apache Ant is the build tool for the Apache NetBeans IDE and must be configured locally. Install Apache Ant
- **JDK 11**: JDK 11 is required to build and run Apache NetBeans

IMPORTANT	If you are building Apache NetBeans on Windows. `Program Files` must be replaced with `PROGRA~1` and `Program Files (x86)` must be replaced with `PROGRA~2` in your `ANT_HOME` environment variable.

Payara Tools Code

The Payara Tools plugin code is split into five modules, these are all located under the `enterprise` directory:

- payara.common
- payara.eecommon
- payara.jakartaee
- payara.micro
- payara.tooling

To test any code changes, recompile the modified module, then rerun the IDE build using `ant tryme`.

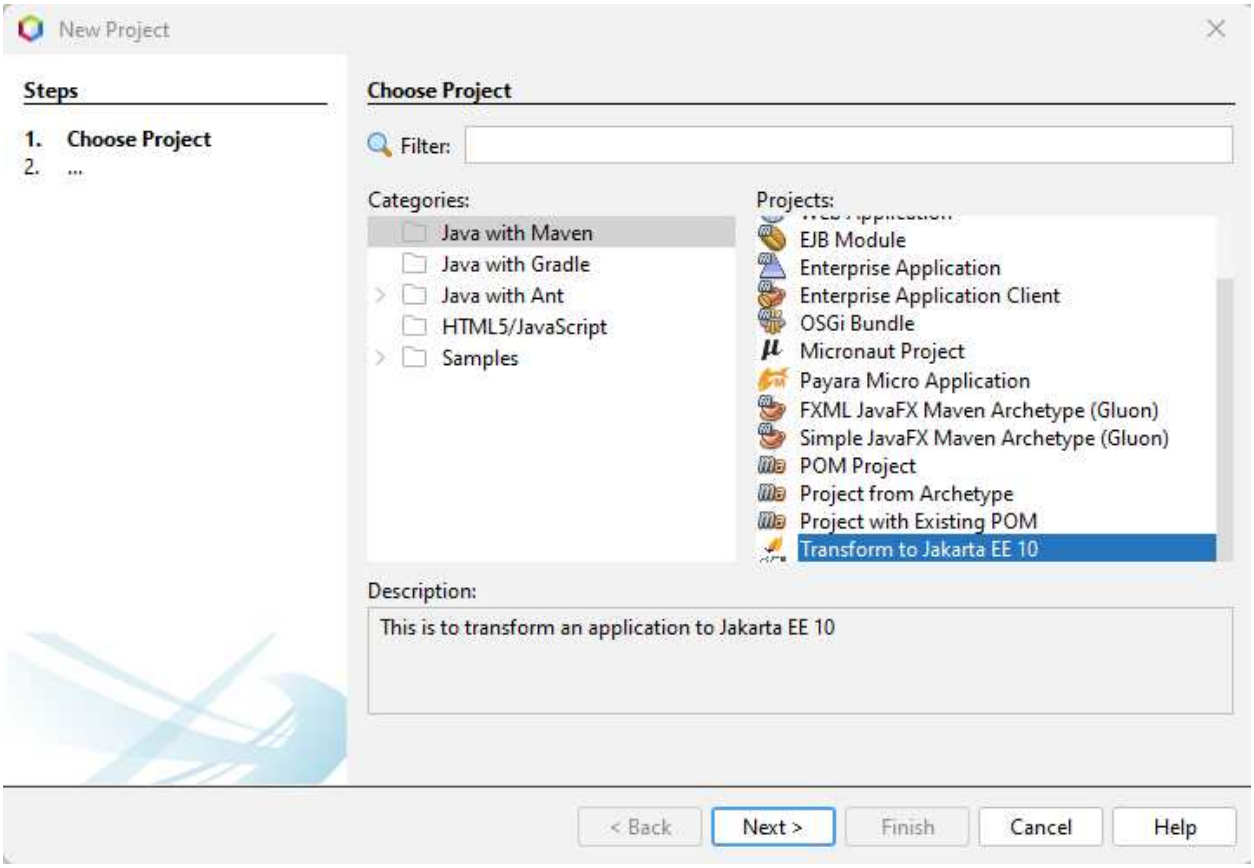
Transform Source Code to Jakarta EE 10

Apache NetBeans IDE provides tools to allow users to automatically convert the source code of Java/Jakarta EE 8 applications to Jakarta EE 10.

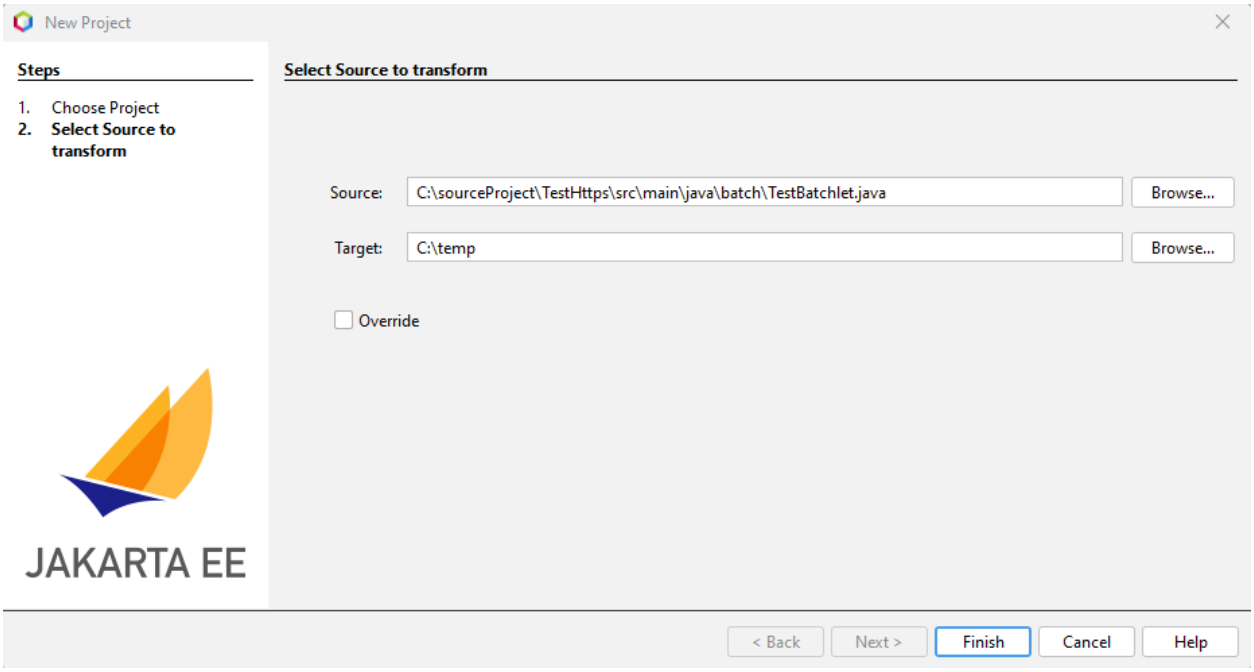
IMPORTANT	The features described on the following page will be available until Apache NetBeans 19 is released. You can see the release dates here: https://cwiki.apache.org/confluence/display/NETBEANS/Release+Schedule

Transform a Single File

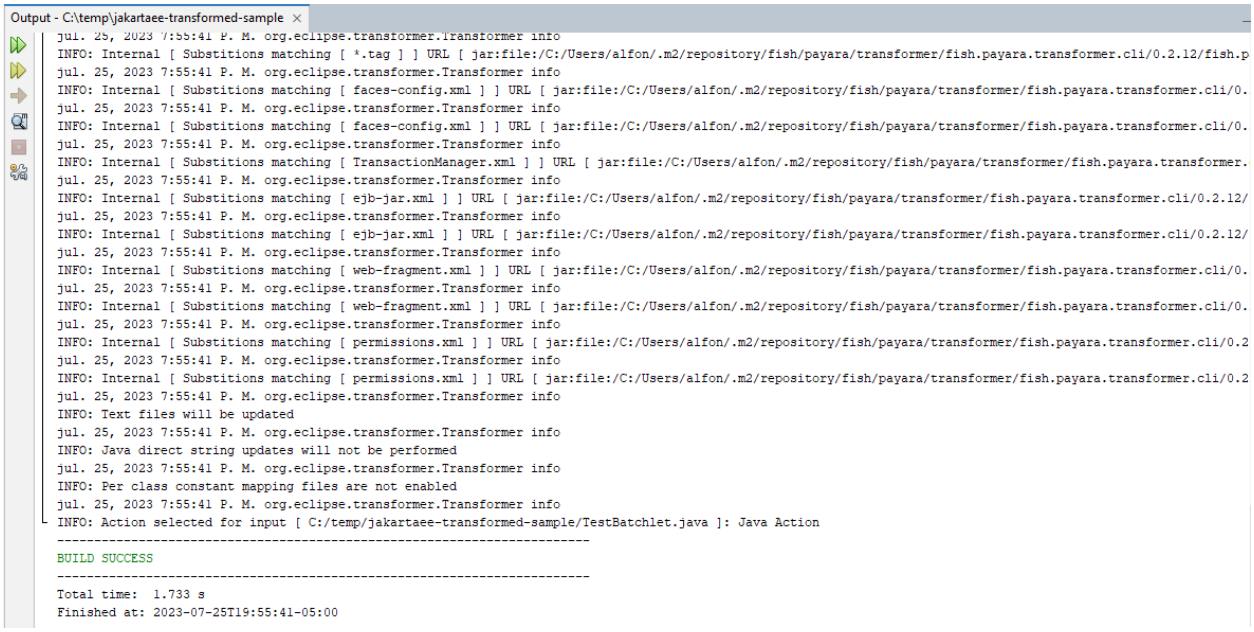
Select from New Project Wizard the option named **Transform to Jakarta EE 10**:



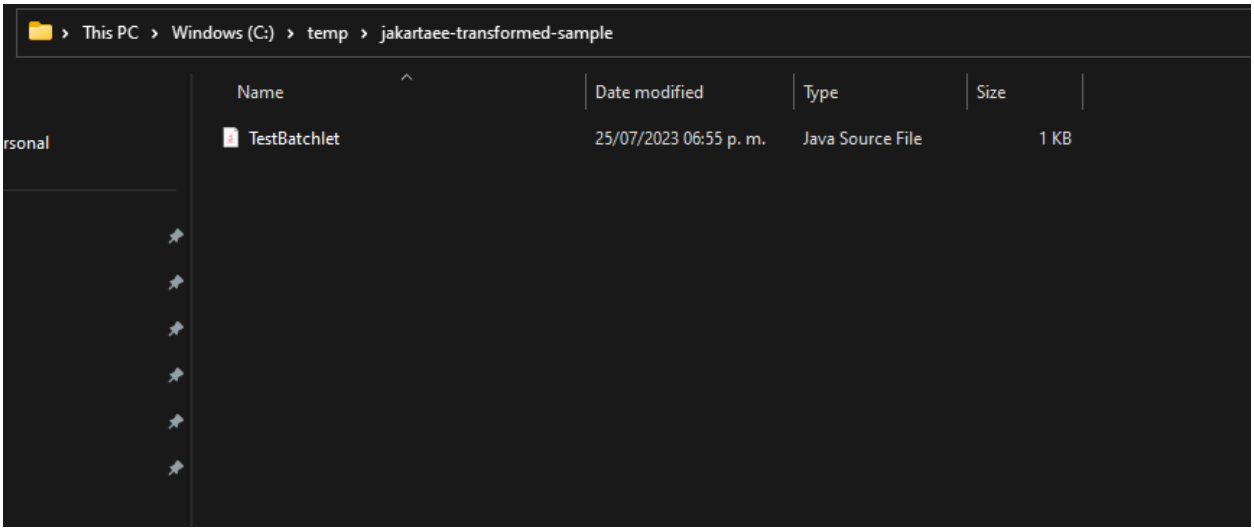
Proceed to select the source file and target folder and click **Finish** to start transformation:



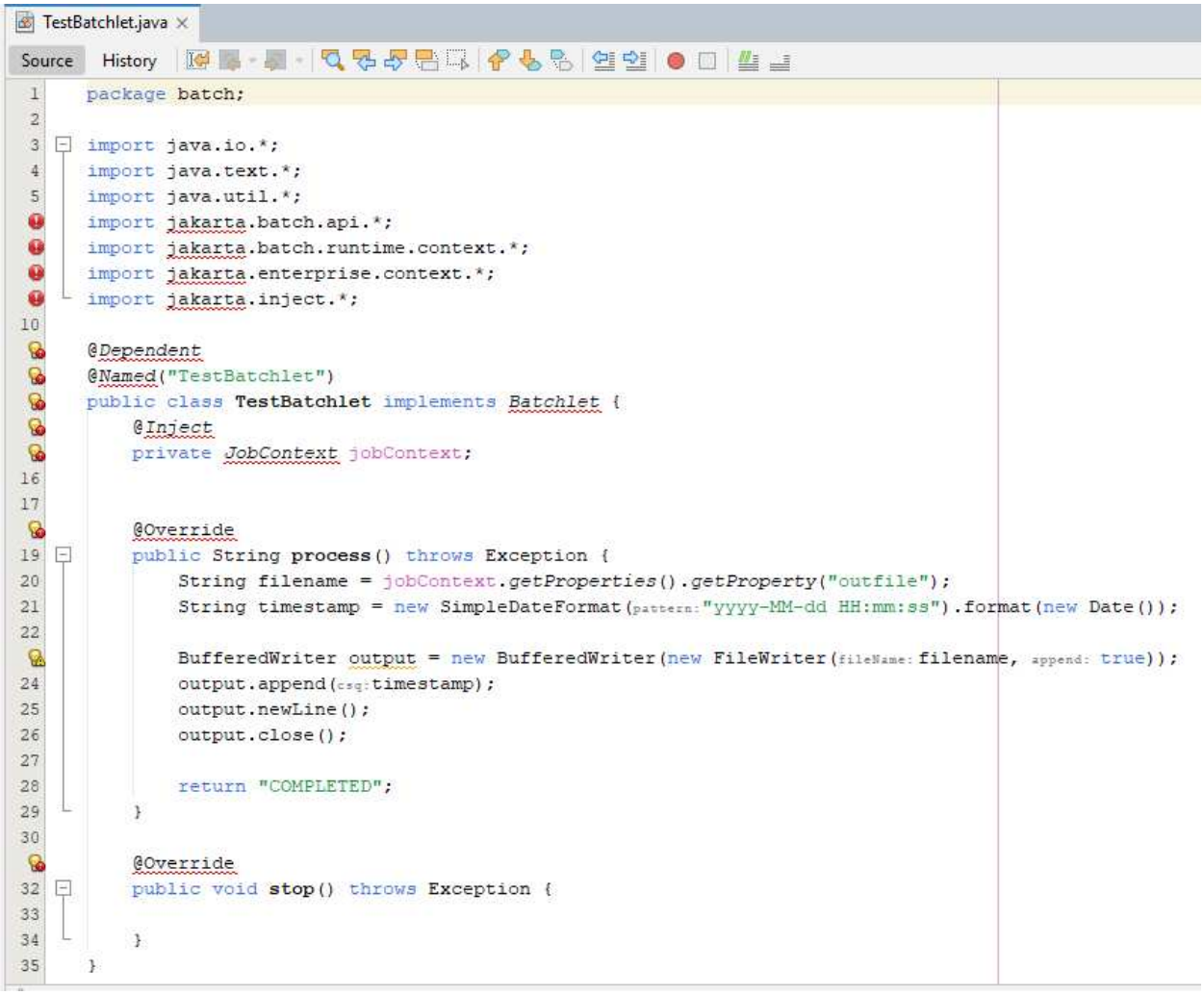
Check the progress and result of the operation on the **Output** tab:



This will create a folder named **jakartaee-transformed-sample** into the target folder (selected from the wizard) with the transformed file:



To verify the resulting file you'll have to open it using Apache NetBeans editor:



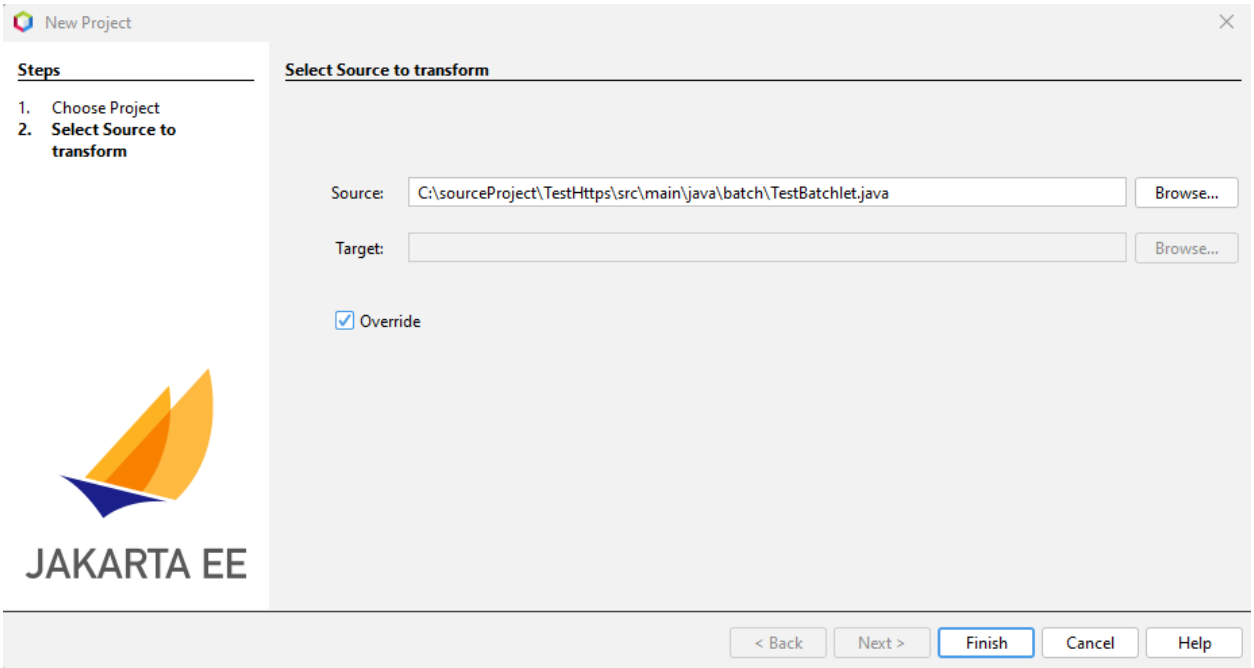
NOTE	From the image you can see errors on the editor. This is happening because after doing the transform operation it is needed to change the

> jakarta dependency manually on the pom file associated with the project.

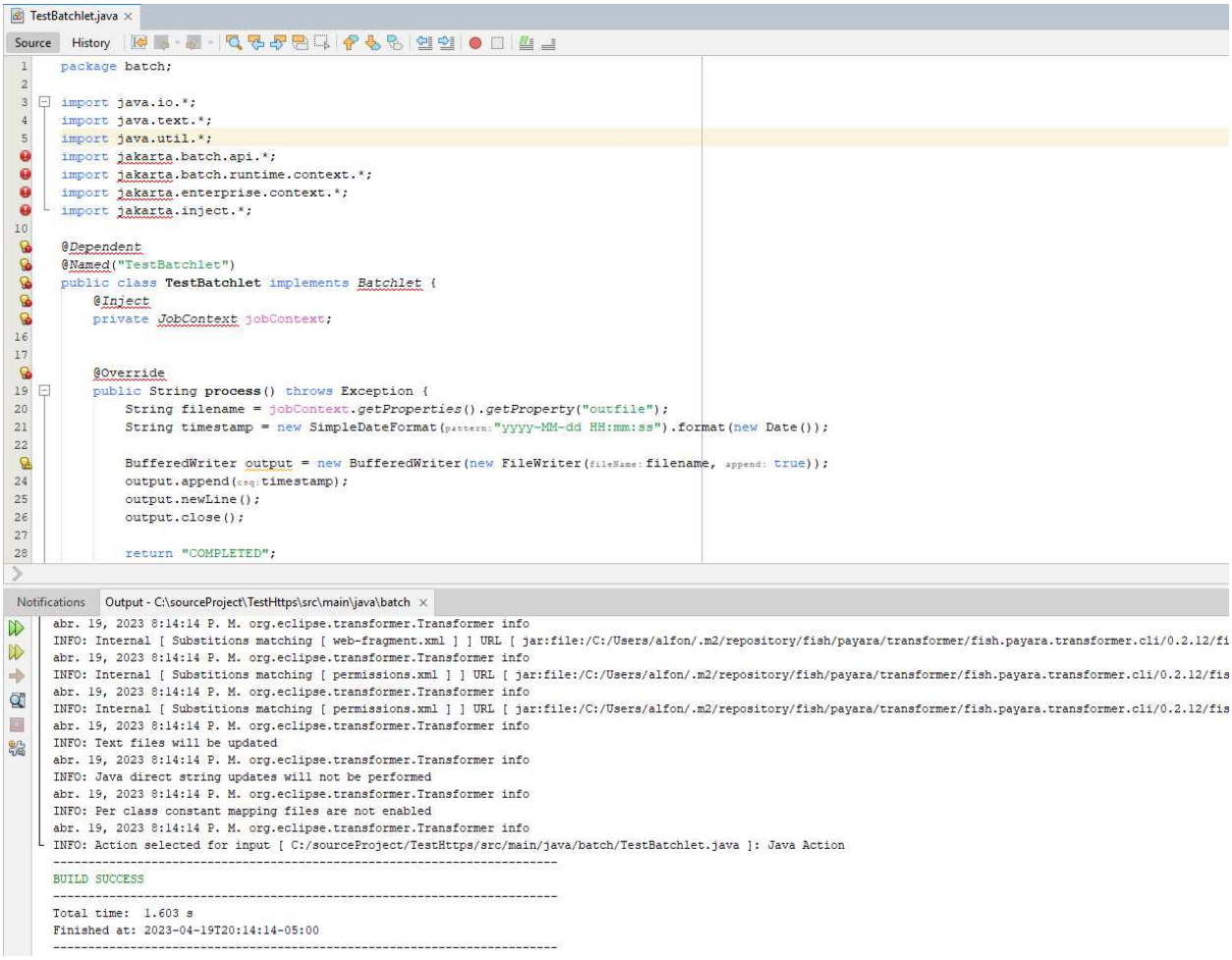
Transform a Single File in the Same Location

It is possible to save the transformed file in the same location where the original resides.

To do this, select the source location and then mark the **Override** option. This will disable and clean the target value, finally select **Finish**:



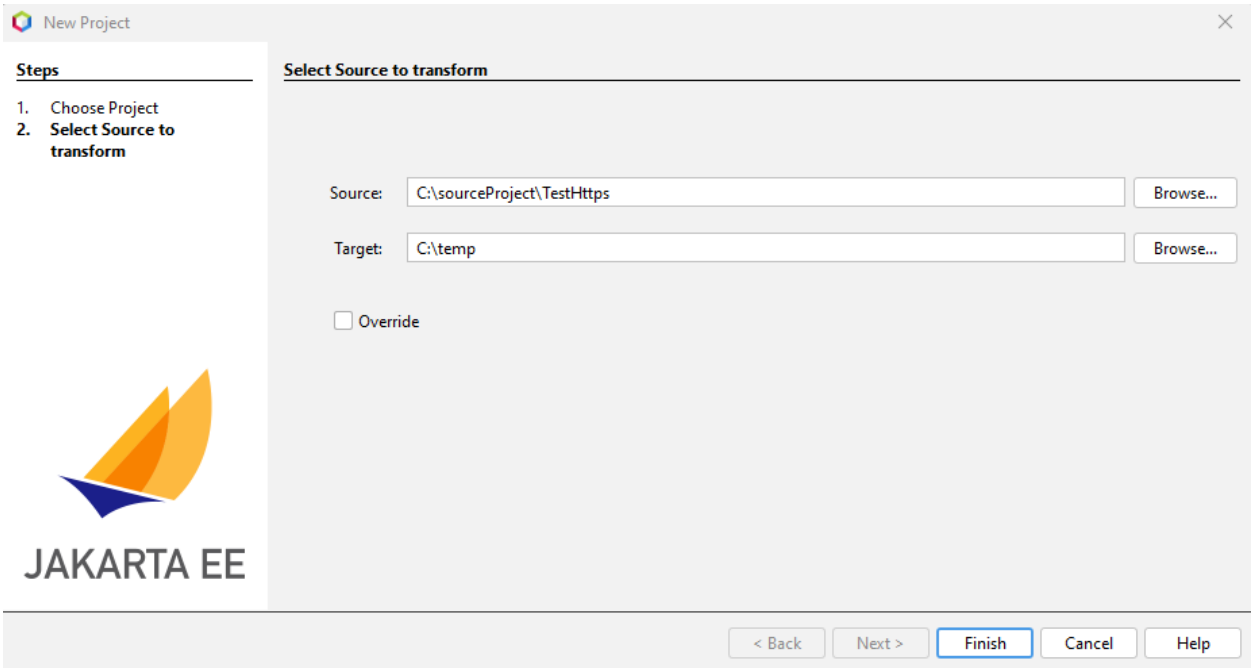
Once the file has been transformed, it will override the file. You can open with NetBeans editor:



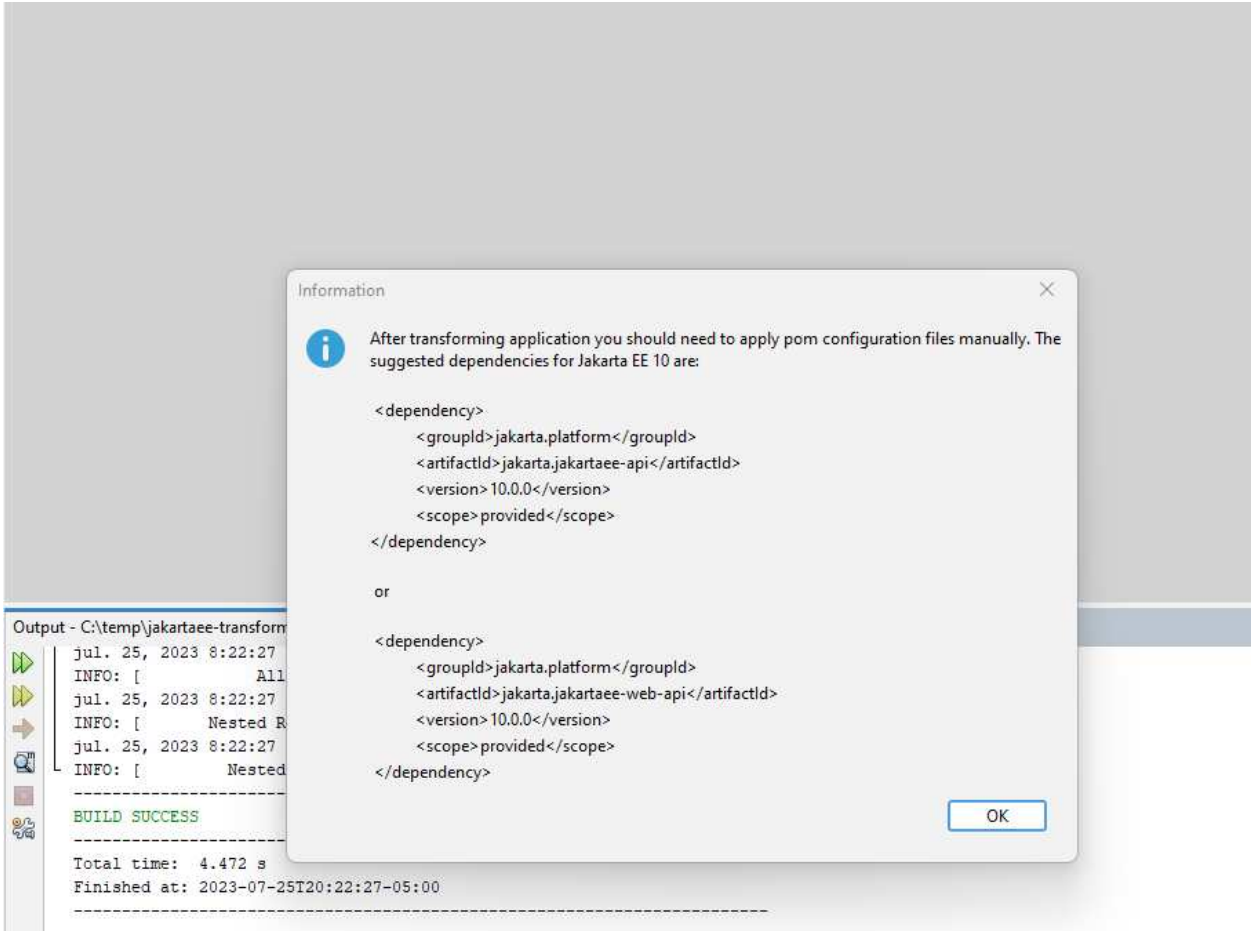
NOTE	From the image you can see errors on the editor. This is happening because after doing the transform operation it is needed to change the jakarta dependency manually on the pom file associated with the project.

Transform All Files in a Folder

You can transform all source code files located in a folder. To do this you must select the source folder and the target folder, finally click **Finish**:



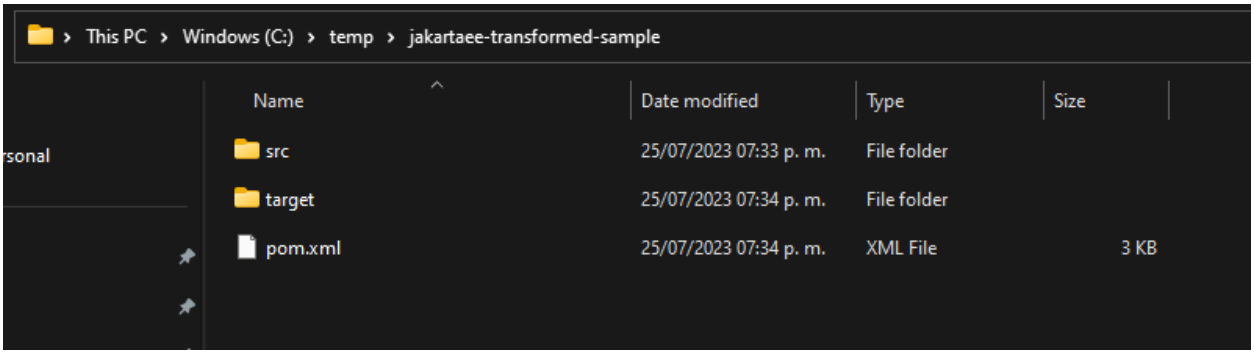
Check the progress and result of the operation on the **Output** tab:



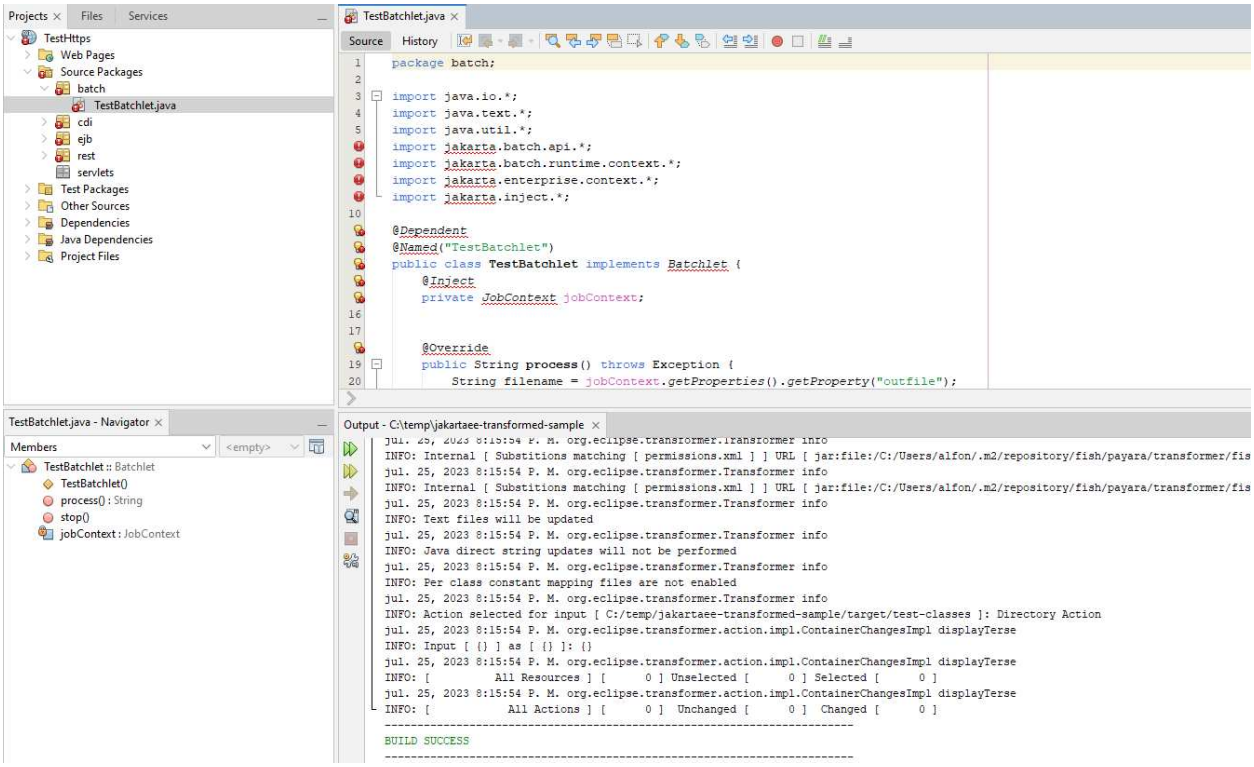
NOTE	The IDE will also show a dialog with additional recommendations to

> configure the source code's project POM configuration. These configuration changes will have to be done manually if required.

This will create a folder named **jakartaee-transformed-sample** into the target folder (selected from the wizard), with the transformed folder and the contents into it:



Apache NetBeans will automatically open the new folder and you can verify results within the IDE:



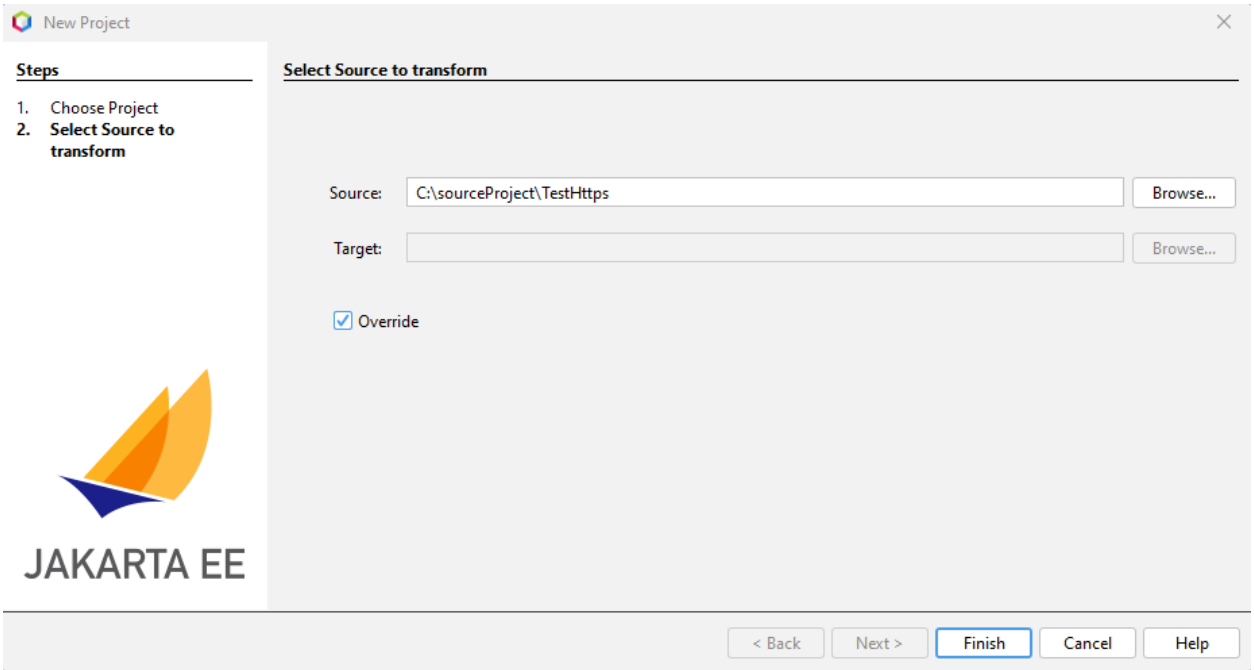
NOTE

> From the image you can see errors on the editor. This is happening because after doing the transform operation it is needed to change the jakarta dependency manually on the pom file associated with the project.

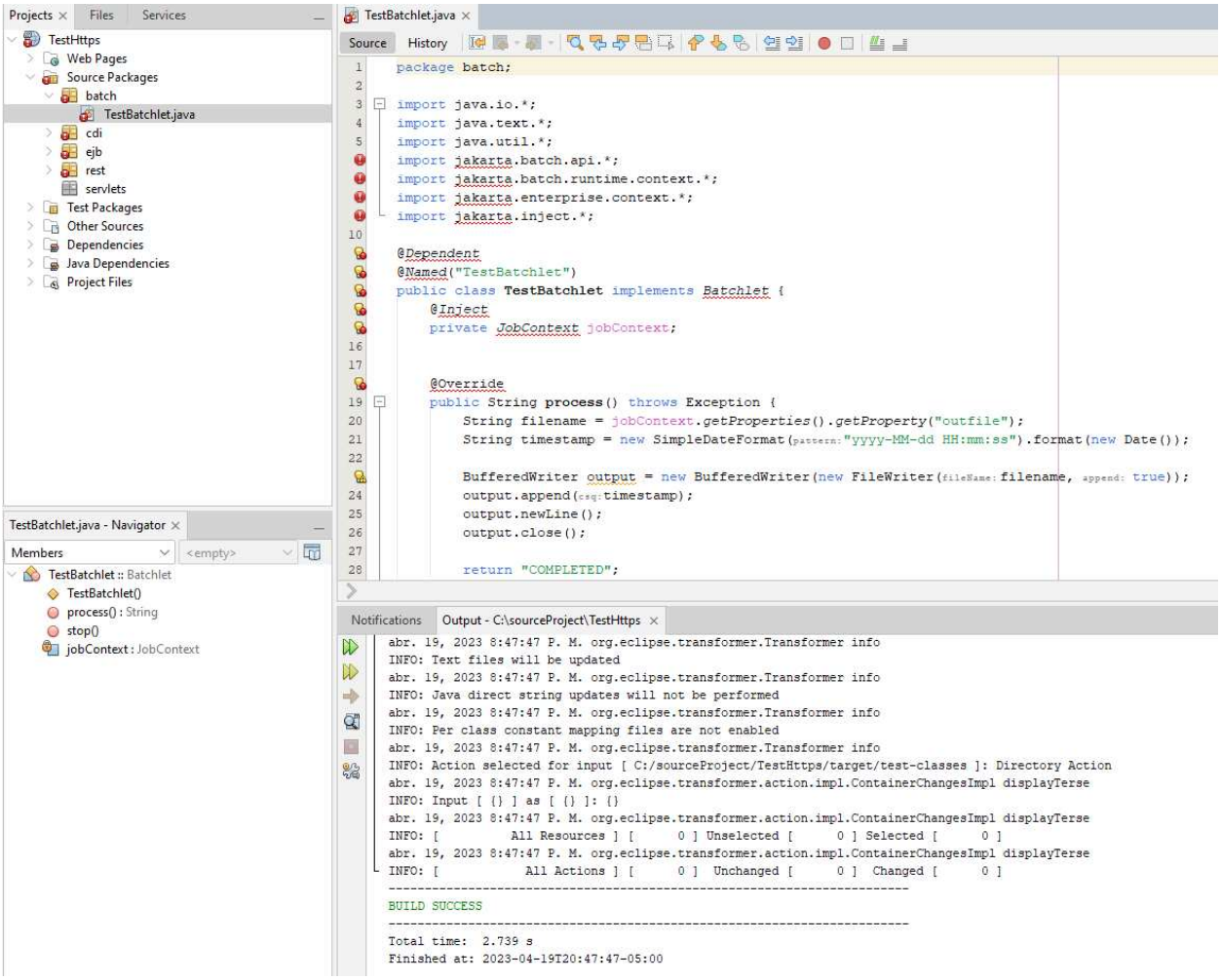
Transform All Files in a Folder in the Same Location

It is possible to save all transformed files in the same location where the originals reside.

To do this you should select the source folder and then mark the **Override** option, finally click **Finish**:



Once the contents of the folder have been transformed, the new files will override the current ones and the IDE will open the project to verify results:



NOTE	From the image you can see errors on the editor. This is happening because after doing the transform operation it is needed to change the jakarta dependency manually on the pom file associated with the project.

VSCode Extension

Payara VS Code Extension

The Payara VS Code extension enables Payara Server and Payara Micro integration with Visual Studio Code. The Payara Tools extension is available in the Visual Studio Marketplace.

Prerequisites for Payara Tools on VS Code

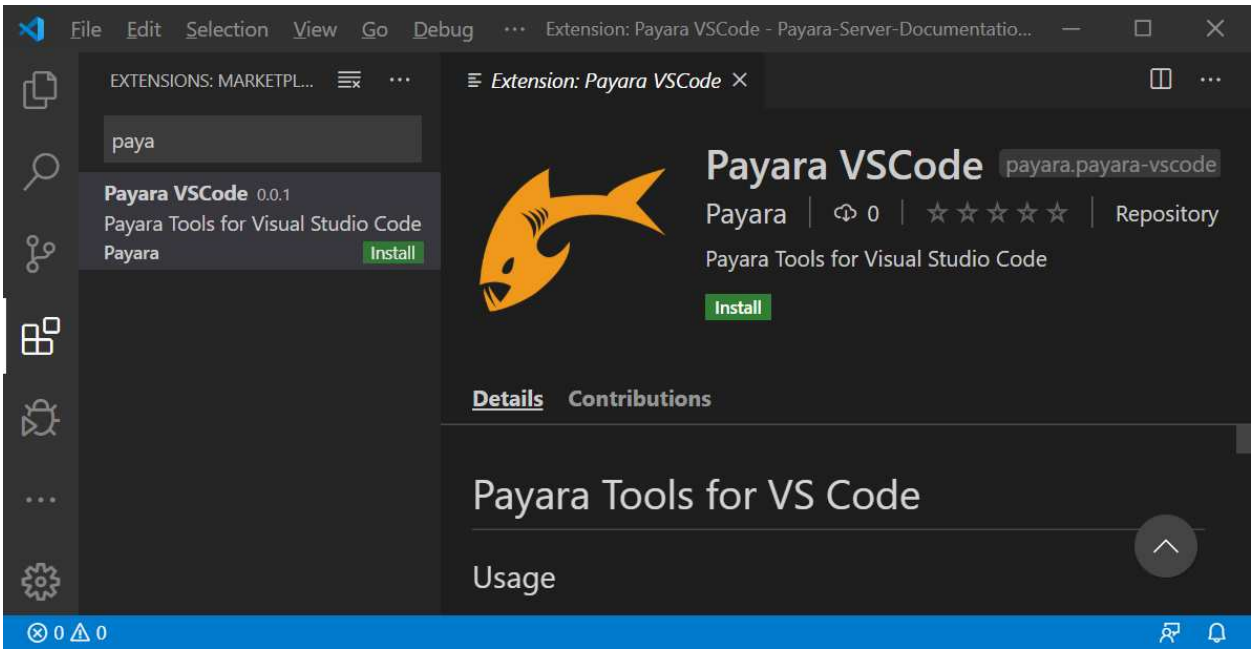
The Java Extension Pack, by Microsoft must be installed first.

IMPORTANT	If you are using Maven or Gradle as the build tool in your project,

> the `MAVEN_HOME` or `GRADLE_HOME` environment variables must be set respectively.

Installing the Payara VS Code Extension

1. Open the **Extensions** panel on the left-most menu of the editor (`Ctrl + Shift + X`)
2. Search for the **Payara Tools** extension.
3. Click the **Install** button.



Azul Payara Tools

- "Payara Server" - The Payara VS Code extension enables developers to easily configure Payara Server and manage applications from Visual Studio Code.
- "Payara Micro" - The Payara VS Code extension provides first-class support to create and run Payara Micro maven web applications from VS Code. Payara Micro is the microservices-ready version of the Payara Server.

Payara Server Tools in VS Code

Adding Payara Server to VS Code

1. Open the command pallet using **Ctrl + Shift + P**, type **Payara** and select the **Add Payara Server** option.

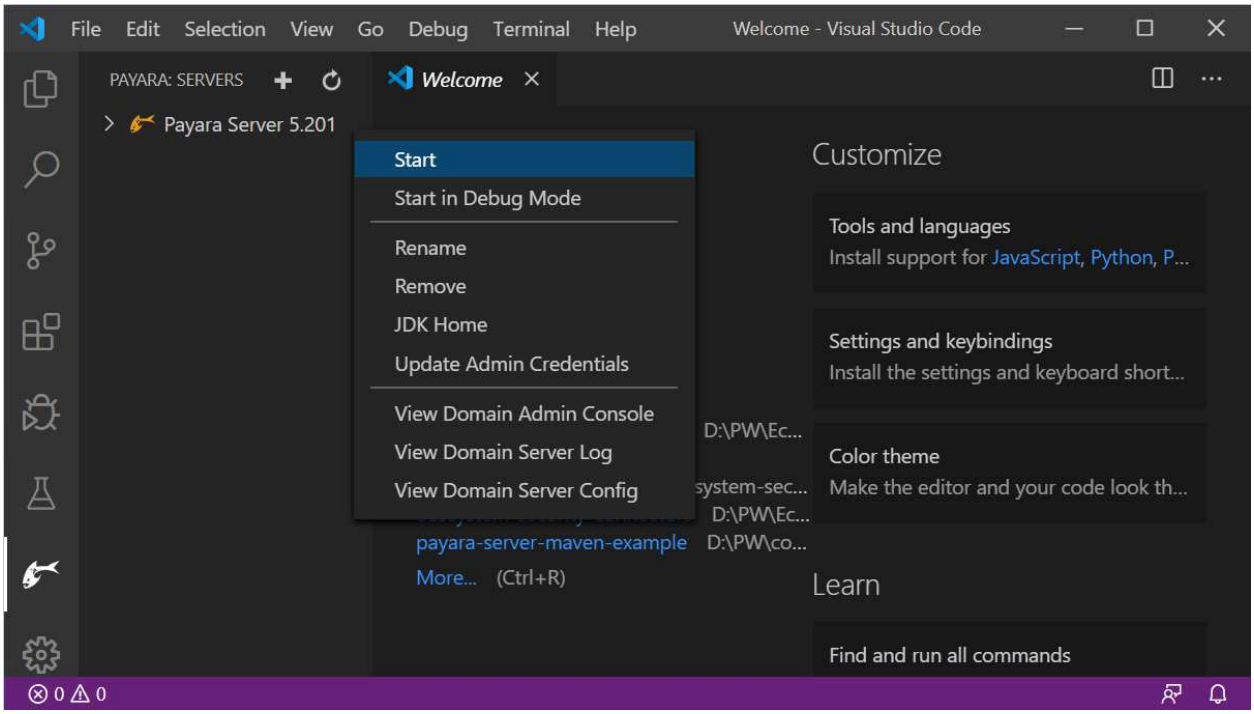
2. Browse the Payara Server installation path (recently used servers location are listed by default).
3. Choose a name for the server.
4. Then select a domain from the list or create a new domain by pressing the **+** button.

[Add Payara Server] | *vscode-extension/payara-server/Add Payara Server VS Code.gif*

Managing Payara Server from VS Code

Once Payara Server is added to VS Code it can be found under the Payara Servers explorer panel with the configured name. Right-clicking on the server lets you perform any of the following actions:

- Start, restart or stop the server
- Start debugging the server
- Rename or remove the server
- Open admin console
- Open server log
- Open server domain.xml config
- Update admin credentials
- Update JDK path

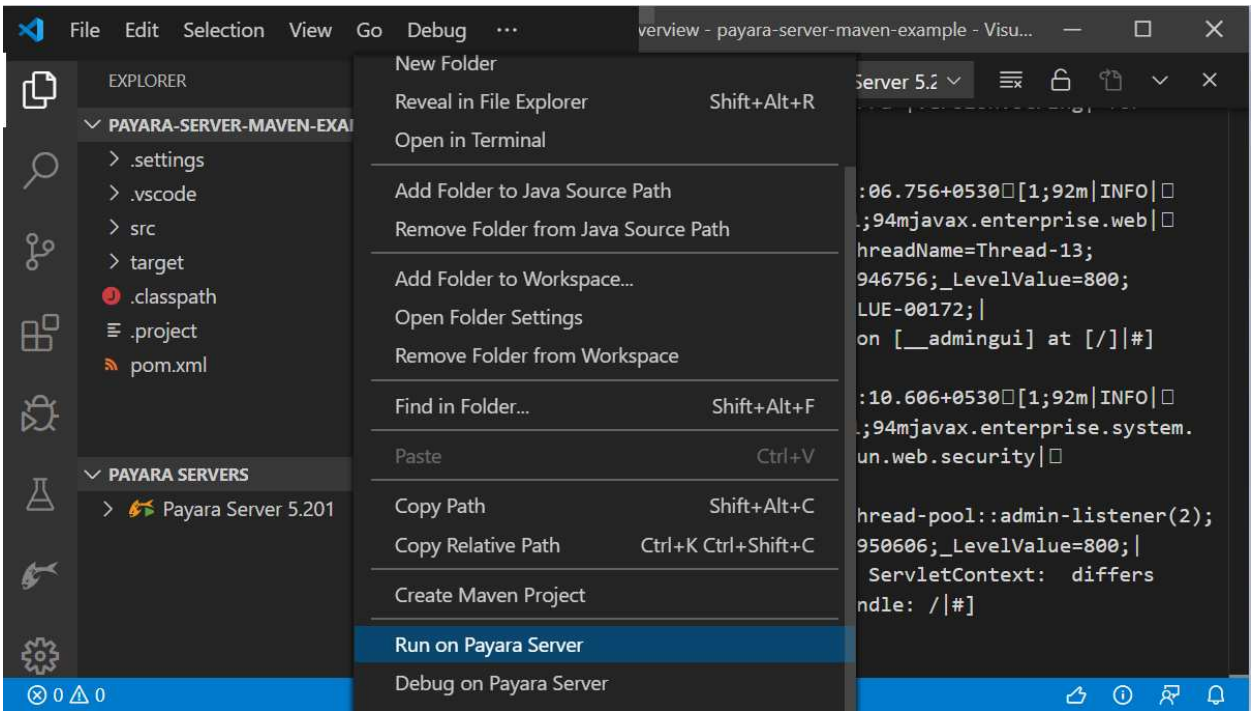


Managing Applications on Payara Server

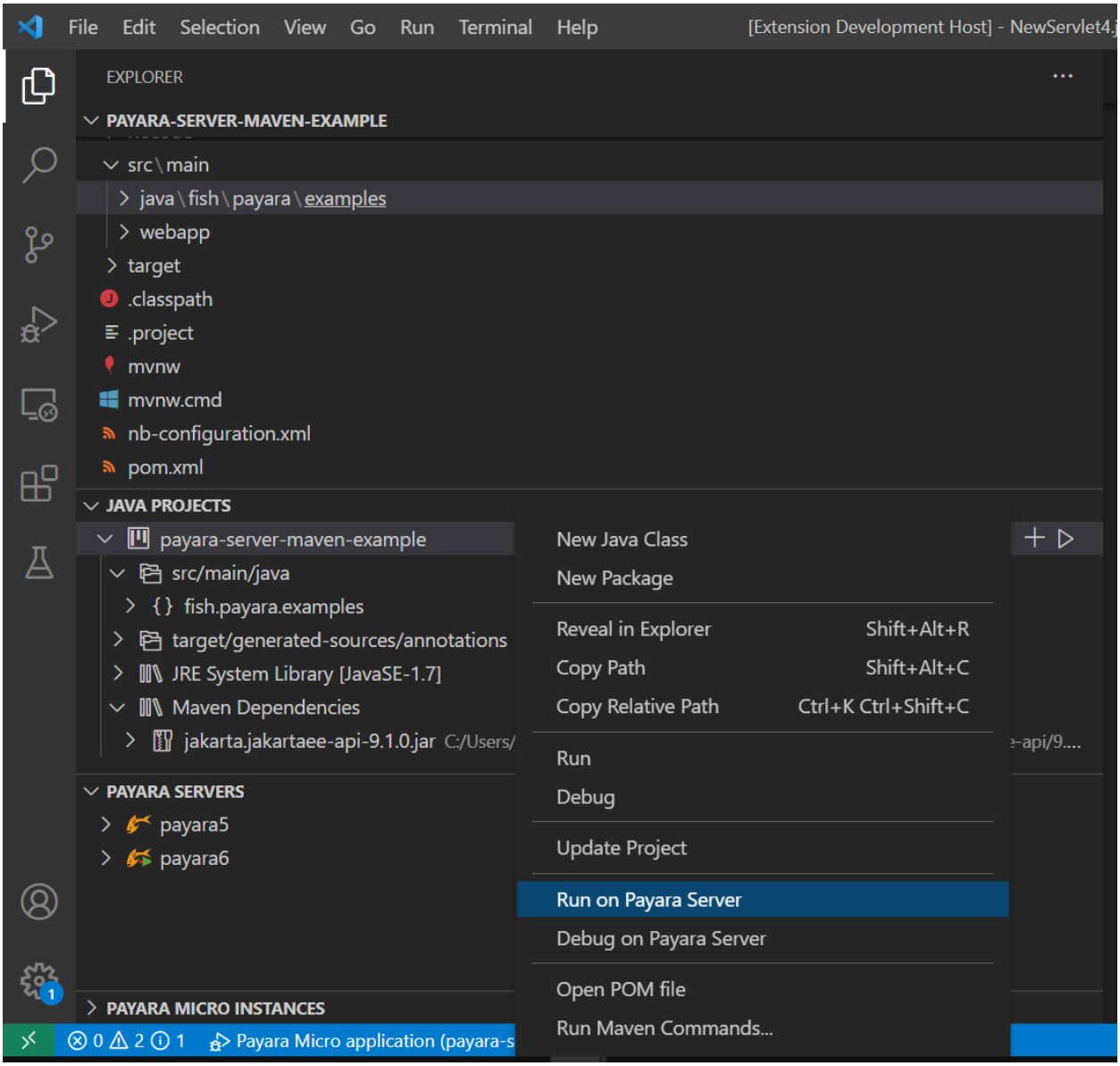
Deploying Applications

An application can be deployed to Payara Server easily though VS Code. Applications can be deployed via either of the following two methods

Right-click on the application and select **Run on Payara Server** to deploy the application:

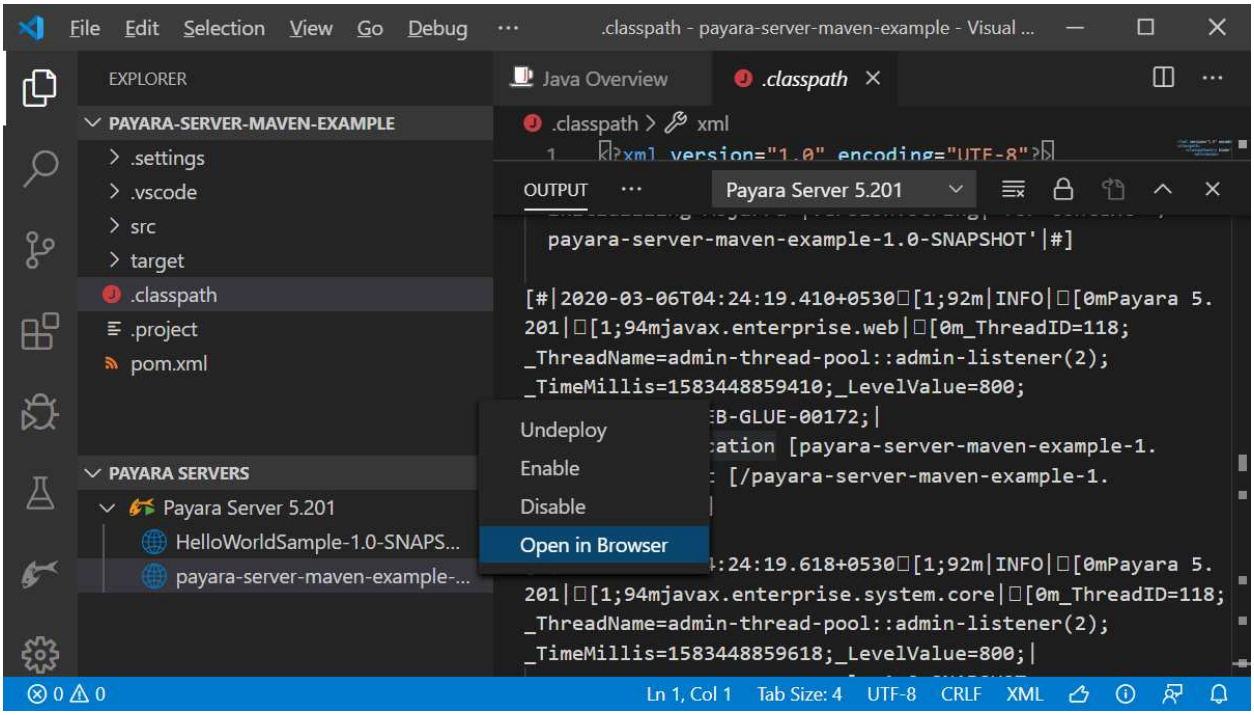


Alternatively, an application can be deployed from the **Java Projects** interface:



Undeploying Applications

Deployed applications can be opened in the browser, undeployed or disabled by right-clicking on the application in the **Payara Servers** interface:



Deploy Applications in a Local Docker Container

The Payara Server tools for the VS Code extension includes Docker integration, making it seamless to deploy exploded applications to a Payara Server instance running within a local Docker container.

To take advantage of this feature, follow these steps below:

1. Register a new Payara Server instance by performing the following:
    a. Open the command pallet using `Ctrl + Shift + P`, type **Payara** and select the **Add Payara Server** option.
    b. Select Remote Domain.
    c. For Docker integration, choose the "Docker" type and provide the `HOST_PATH` (e.g., `D:/test`) and `CONTAINER_PATH` (e.g., `/mnt/test`) mapping details.

    This mapping should align with the exploded application path or parent path of exploded application.

    d. Choose a name for the server.
    e. Enter the domain name, host name, and necessary credentials.

2. Run the Payara Server Docker container using the following command:

```
docker run -p 8080:8080 -p 4848:4848 -p 9009:9009 -v HOST_PATH:CONTAINER_PATH
payara/server-full:{release-id}
```

   Replace `HOST_PATH` with the actual host path and `CONTAINER_PATH` with the desired container path. The `HOST_PATH` points to the location on your host machine, and `CONTAINER_PATH` is the corresponding location inside the container.

3. Create your web application and deploy it to the registered Payara Server instance.

Deploy Applications in a WSL Environment

Another noteworthy feature is the integration of the Payara Server Tools with Windows Subsystem for Linux (WSL).

This integration empowers developers to manage and deploy exploded applications to a Payara Server instance running within WSL.

Here's how you can make use of this feature:

1. Start the Payara Server instance within WSL:

   a. Run the following command to start the server:

```
./asadmin start-domain
```

   b. Set a password for the admin user:

```
./asadmin change-admin-password
```

   c. Enable secure admin:

```
./asadmin enable-secure-admin
```

   d. Restart the Payara Server domain:

```
./asadmin restart-domain
```

e. Fetch the IP address of WSL environment using the command `hostname -I` and enter it in the `host name` field in the VS Code settings.

2. Register the WSL-based Payara Server instance in VS Code:

   a. Open the command pallet using `Ctrl + Shift + P`, type **Payara** and select the **Add Payara Server** option.

   b. Select Remote Domain.

   c. For WSL integration, choose the `WSL` instance type.

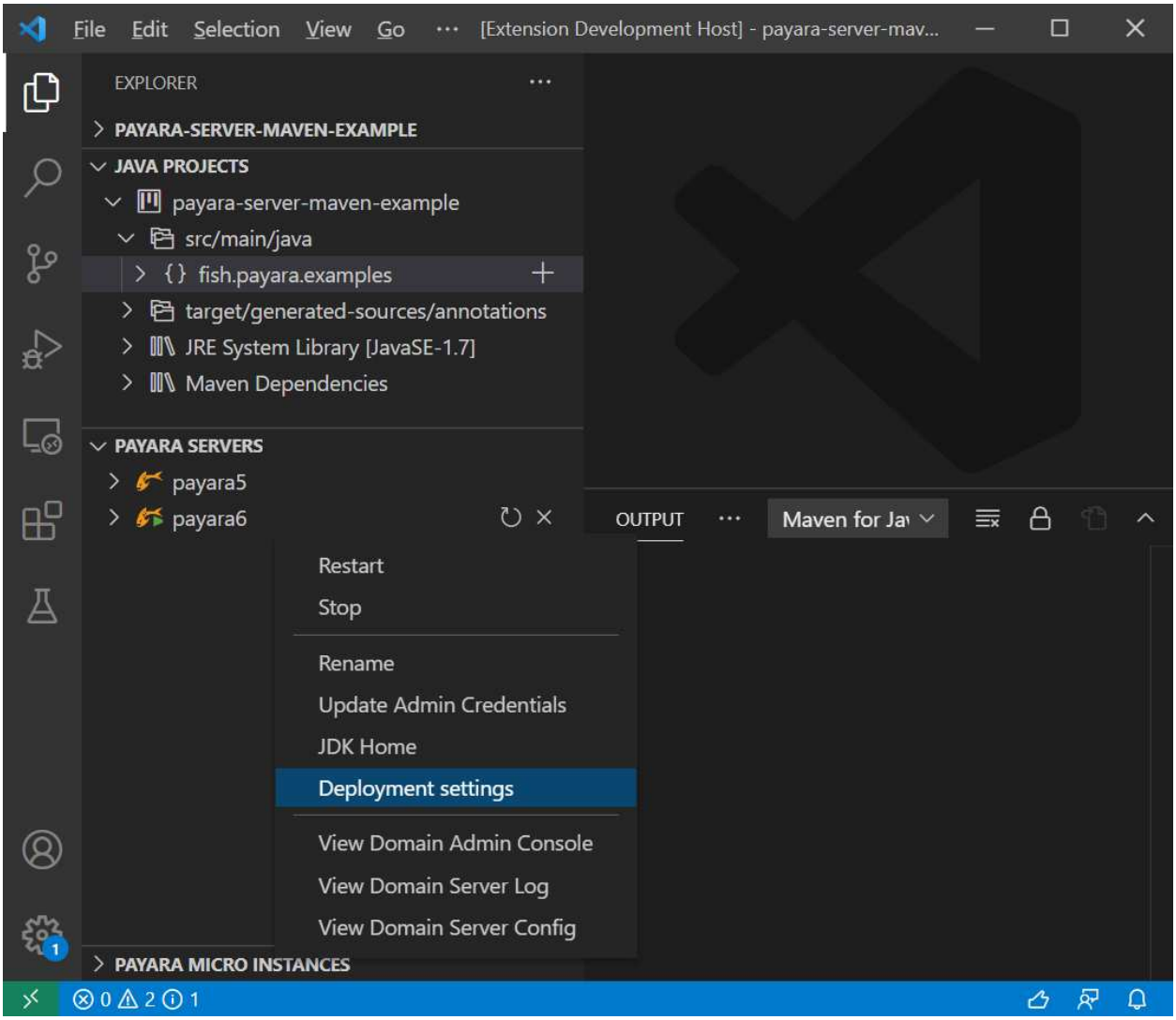
   d. Choose a name for the server.

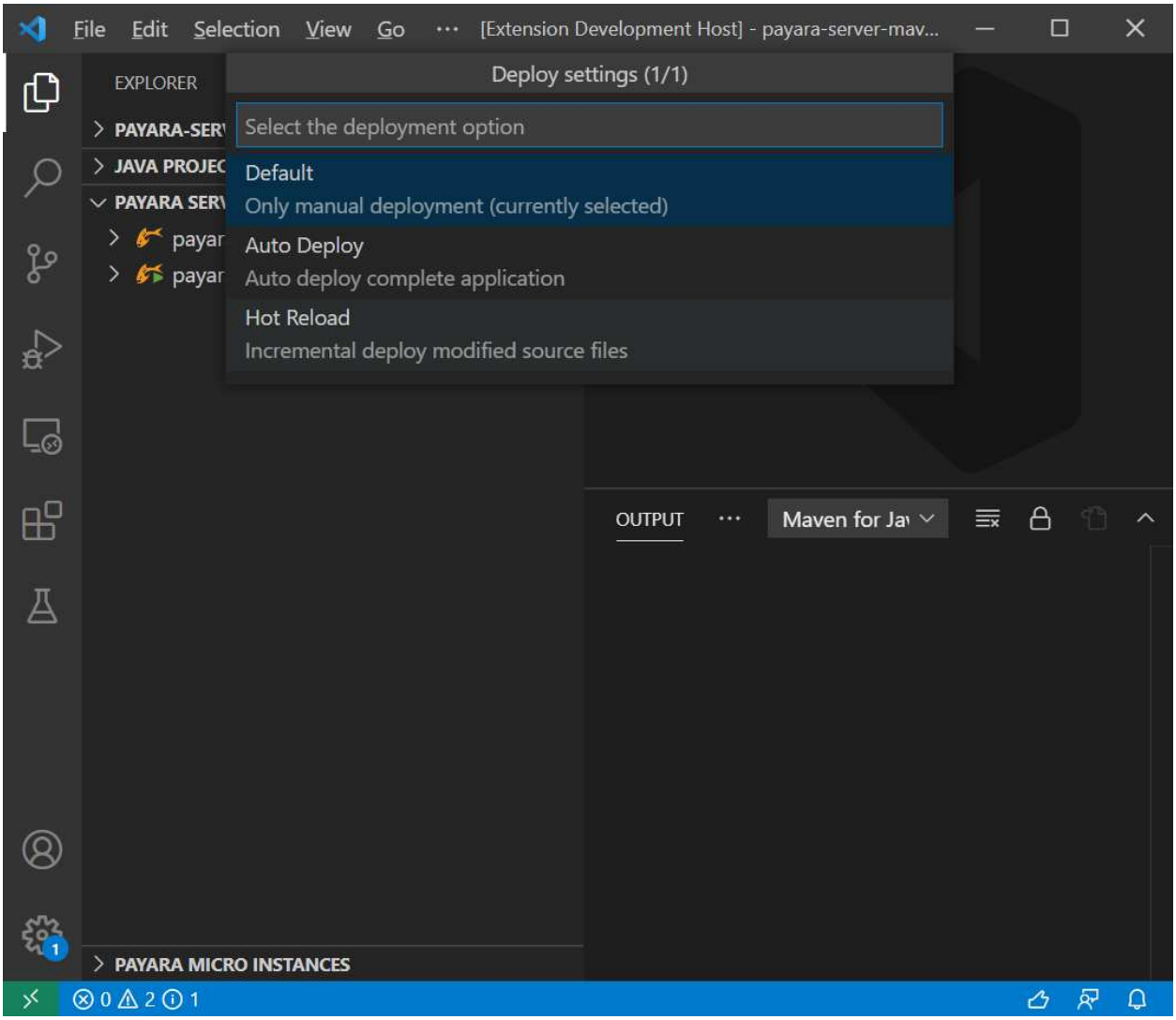
   e. Enter the domain name, host name (fetched via the command `hostname -I` in WSL instance), and necessary credentials.

3. Create your web application and deploy it to the registered Payara Server instance.

Hot Deploy and Auto Deploy Settings

Hot Deploy and **Auto Deploy** can be enabled in Visual Studio Code IDE from the **Deployment Settings** of Payara Server:


File Edit Selection View Go ··· [Extension Development Host] - payara-server-mav...
EXPLORER
PAYARA-SERVER-MAVEN-EXAMPLE
JAVA PROJECTS
payara-server-maven-example
src/main/java
{} fish.payara.examples
target/generated-sources/annotations
JRE System Library [JavaSE-1.7]
Maven Dependencies
PAYARA SERVERS
payara5
payara6
OUTPUT
Maven for Ja
Restart
Stop
Rename
Update Admin Credentials
JDK Home
Deployment settings
View Domain Admin Console
View Domain Server Log
View Domain Server Config
PAYARA MICRO INSTANCES
0 2 1



See Also

"Hot Deploy and Auto Deploy"

Payara Micro Tools in VS Code

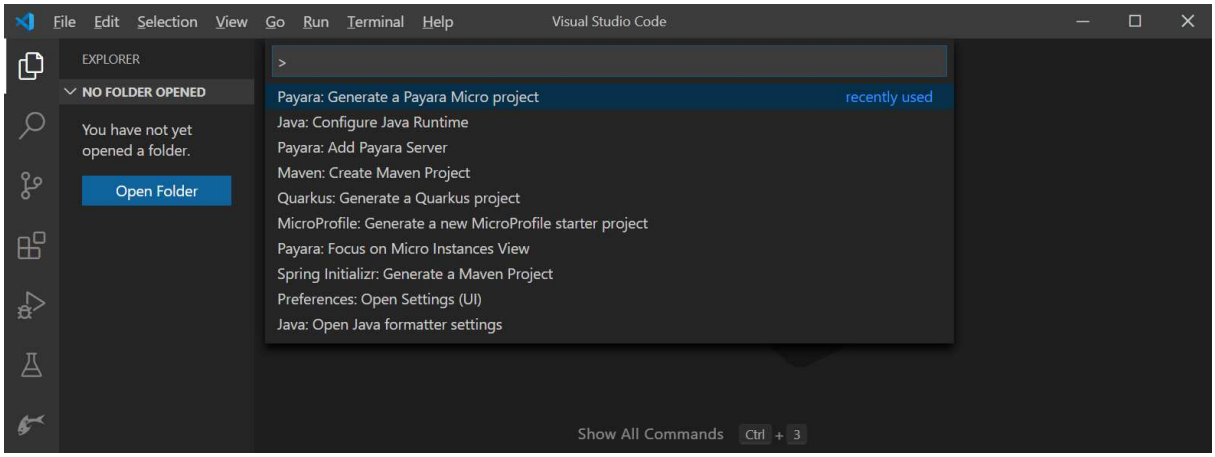
Using the Payara Tools VS Code extension, you can integrate Payara Micro within the VS Code IDE to create new Payara Micro Maven applications, start, stop and deploy applications to Payara Micro.

The extension also allows you to use the Hot Deploy and AutoDeploy features for a better developer experience.

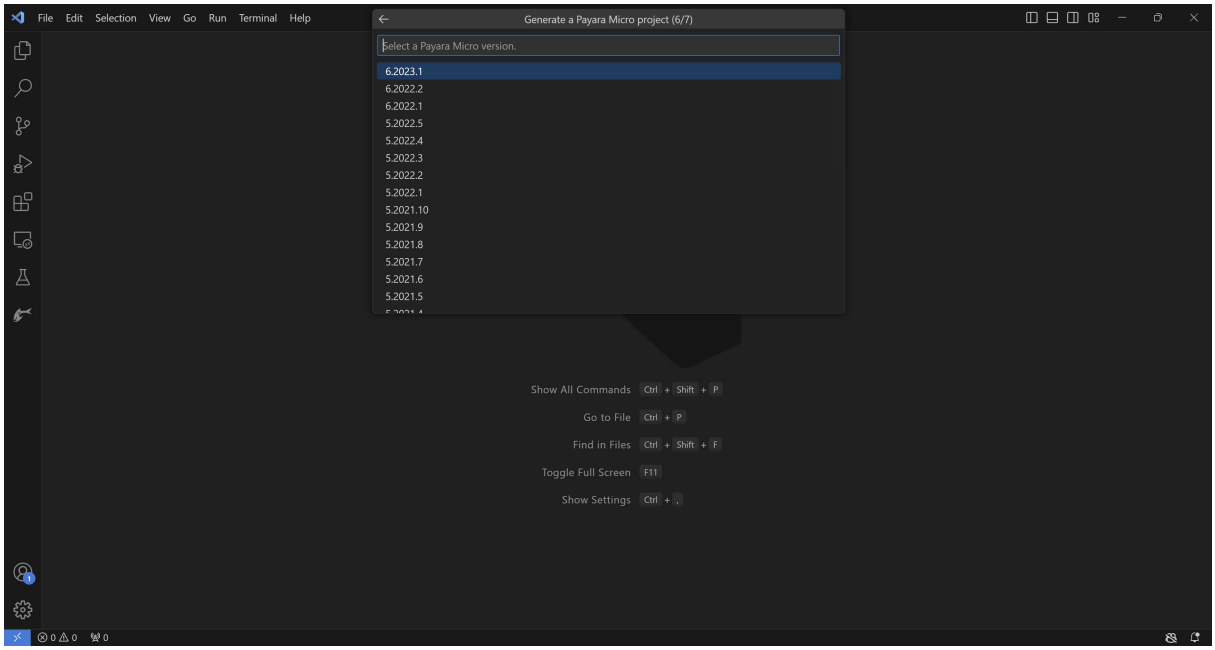
Creating New Payara Micro Maven Application

1. Open the command pallet using `Ctrl + Shift + P`.

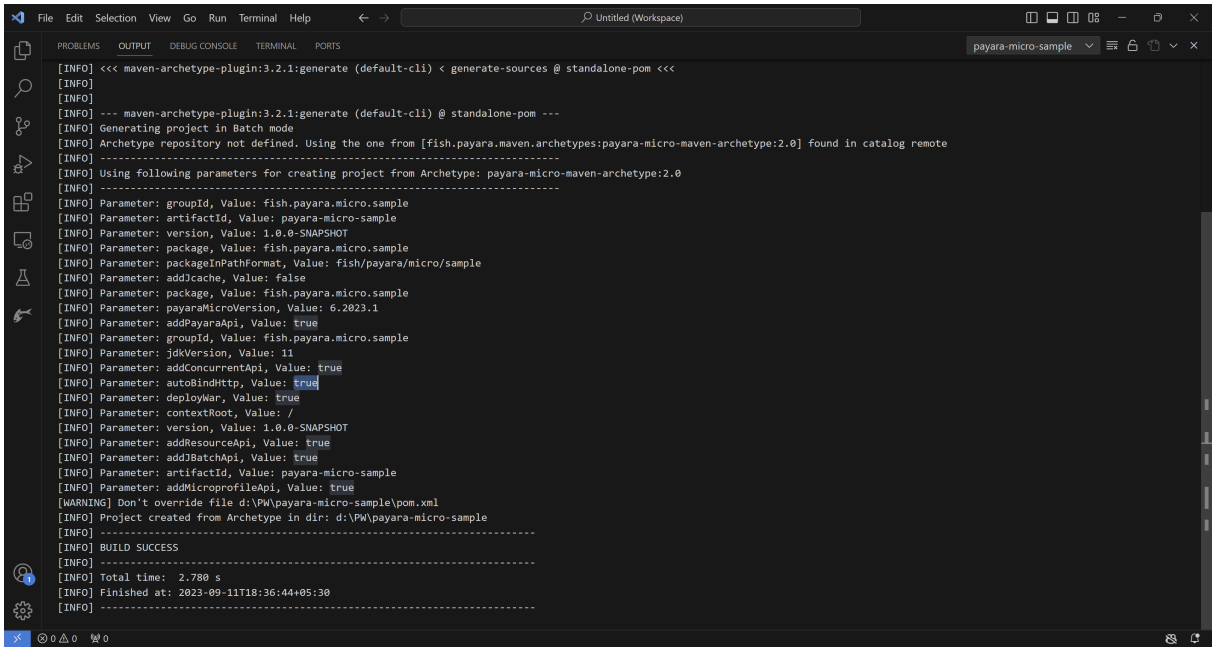
2. Search **Generate a Payara Micro project** and select the available option:



3. Enter the `groupId`, `artifactId` and `version` to setup the project
4. Select the Payara Micro version and the destination folder, where the project will be generated:



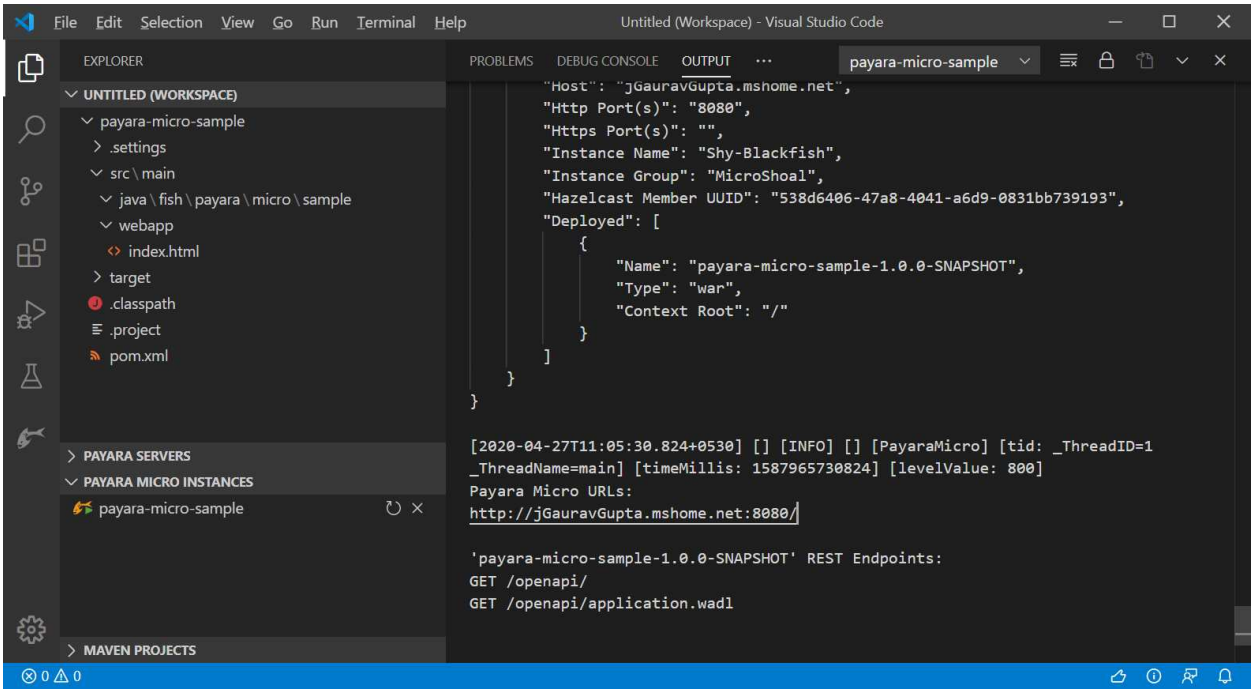
5. VS Code creates the new project from the Payara Micro Maven archetype. You may add the generated project to the current workspace or open in the new window:



Run Payara Micro Application

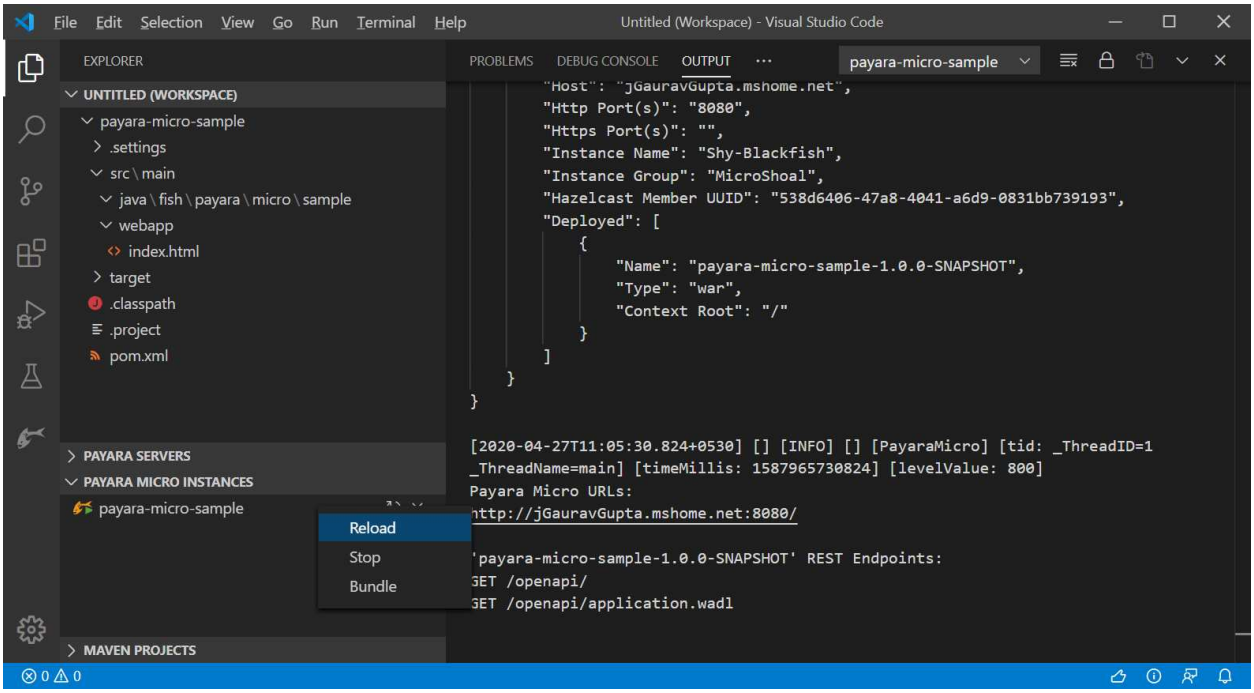
To run the Payara Micro application from the VS Code IDE, go to the explorer sidebar or Payara activity bar, expand Payara Micro Instances, right-click on the listed Payara Micro instance and select **Start**.

After starting the Micro instance, you can view the server logs in the *OUTPUT* window:



To redeploy the application, right-click on the listed Payara Micro instance and select

Reload.



See Also

"Hot Deploy and Auto Deploy"

Building Payara Tools VS Code IDE Plugin

The Payara Tools VS Code IDE Extension source code is available in the VS Code Extension repository.

Prerequisites

Before building the Payara Tools VS Code extension, you will need to configure your local environment. Subsequent builds will not require these steps.

- **Clone the ecosystem-vscode-plugin repository**: You must first clone the repository where the extension source code is stored. VS Code Extension repository
- **Install Yarn**: Yarn is the build tool for the Payara Tools VS Code IDE Plugin and must be configured locally. Install Yarn
- **Install vsce**: vsce is the tool to package VS Code extensions. Install vsce

Build Payara Tools

To build the Payara Tools VS Code extension, run the following commands from the root directory where the project was cloned:

1. `yarn install`
2. `yarn run tslint`
3. `yarn run compile`

To package the extension into a file which can be added to VS Code, run the following command:

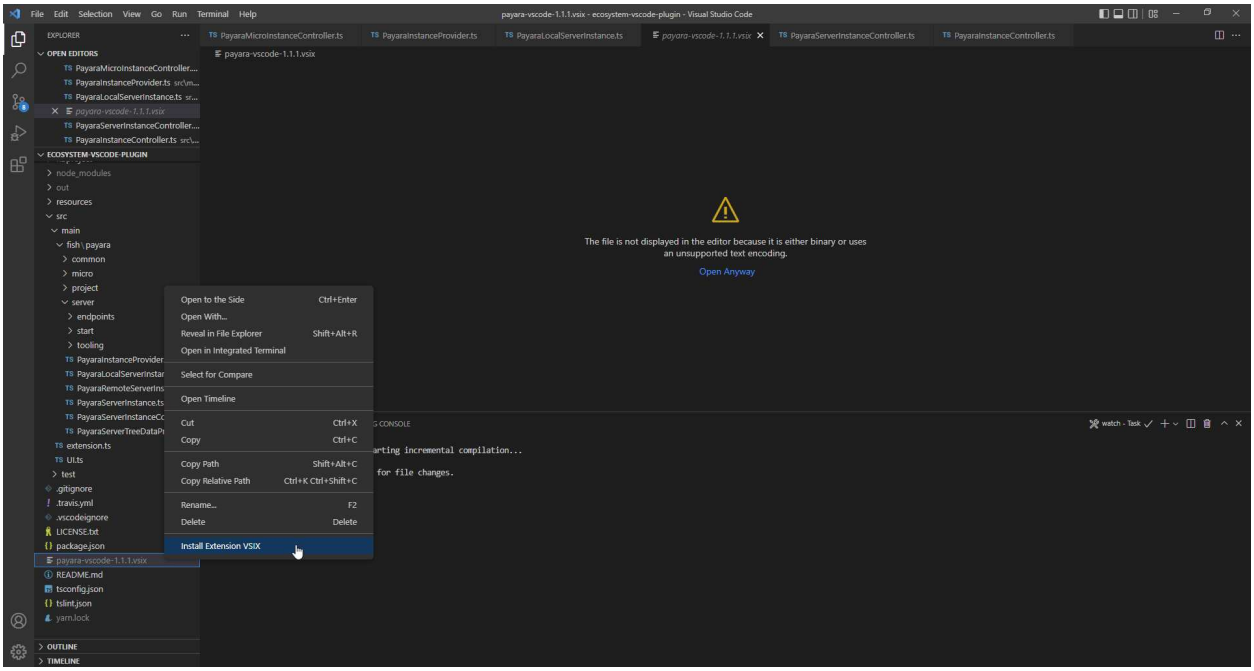
1. `vsce package`

In the root directory of the project there will be a file `payara-vscode-{version-number}.vsix` which will be used to install into VS Code.

Install Payara Tools into VS Code

After Payara Tools for VS Code has been compiled and packaged successfully, you need to add it into the VS Code IDE.

Installing a vsix file in VS Code is as simple as right clicking the file in the explorer, and selecting `Install Extension VSIX`

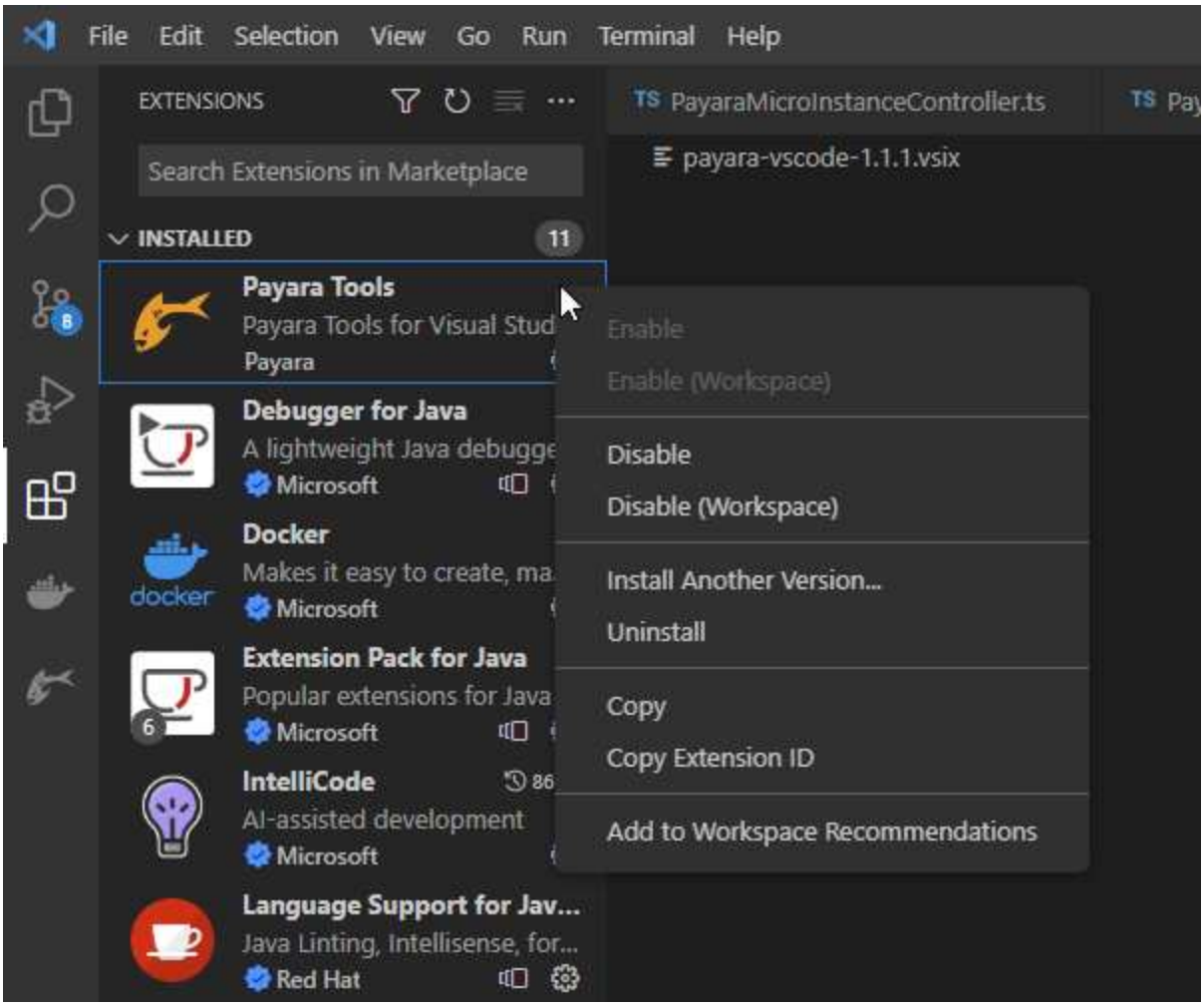


Removing Payara Tools from VS Code

Removing Payara Tools from VS Code after installing using the method above is the same as removing any extension from VS Code.

Removing Payara Tools from VS Code

1. Navigate to Extensions (Ctrl + Shift + X)
2. Right click `Payara Tools` and click Uninstall

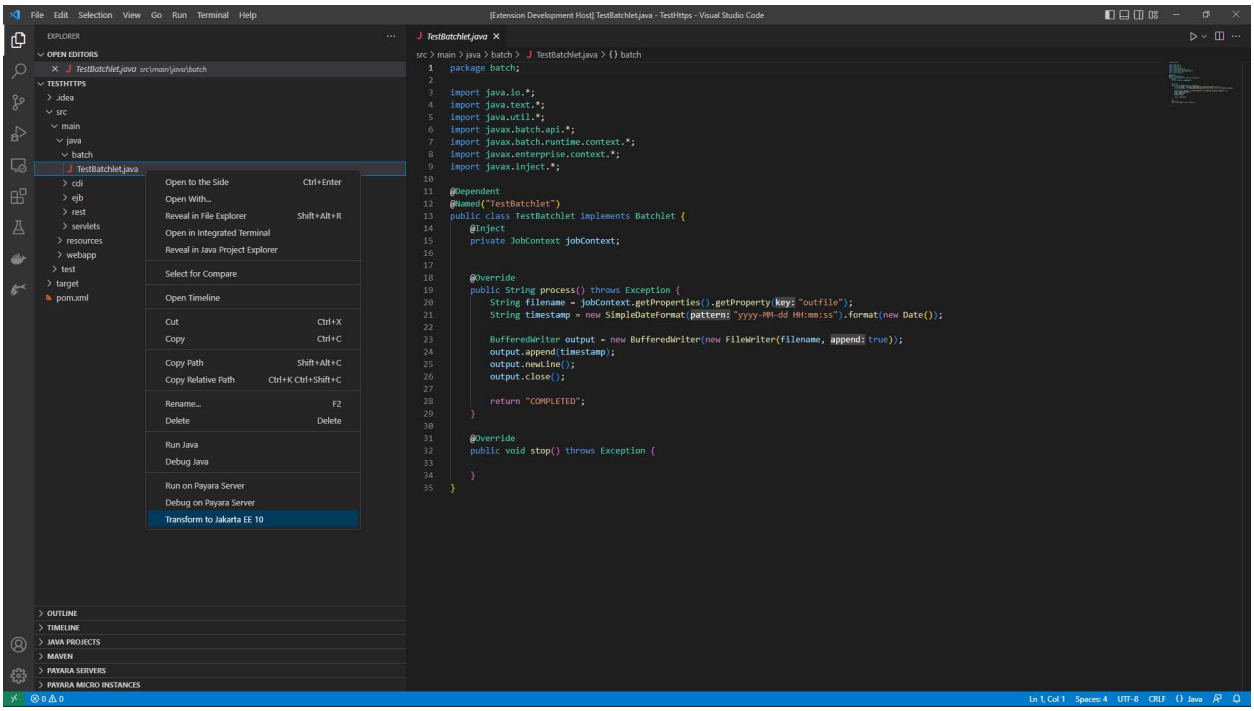


Transform Source Code to Jakarta EE 10

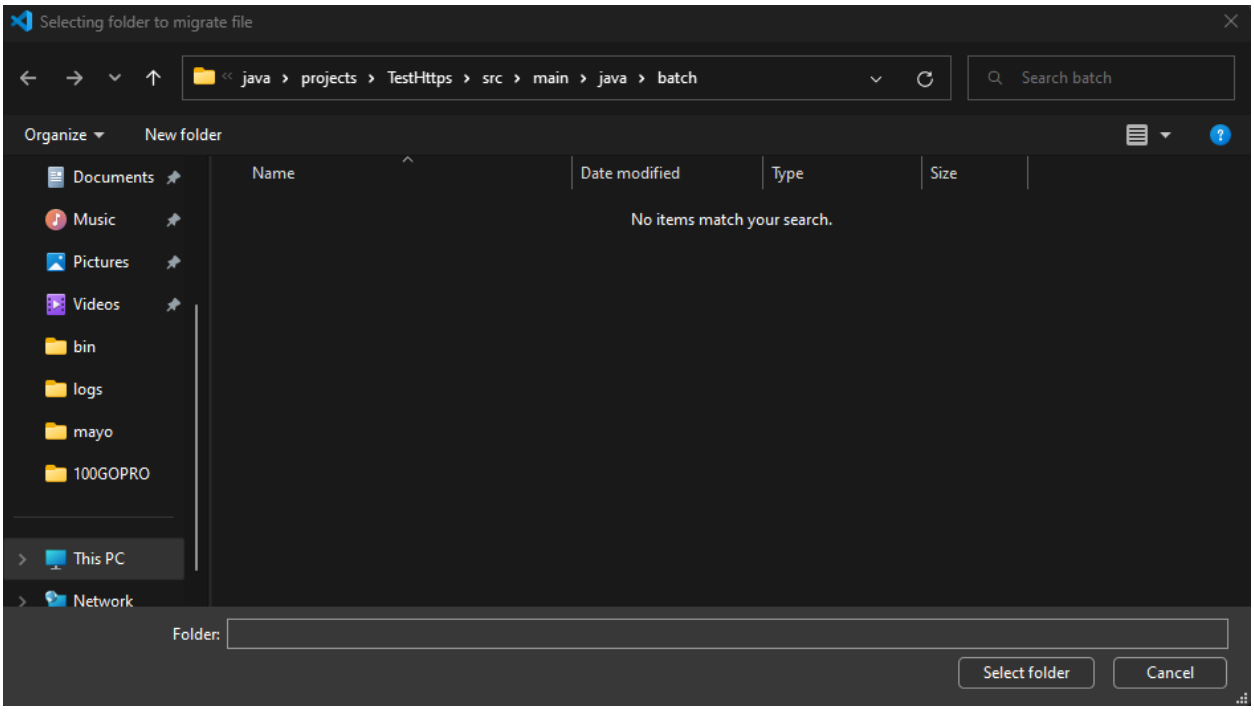
VSCode's Payara Tools allow users to automatically convert the source code of Java/Jakarta EE 8 applications to Jakarta EE 10.

Transform a Single File

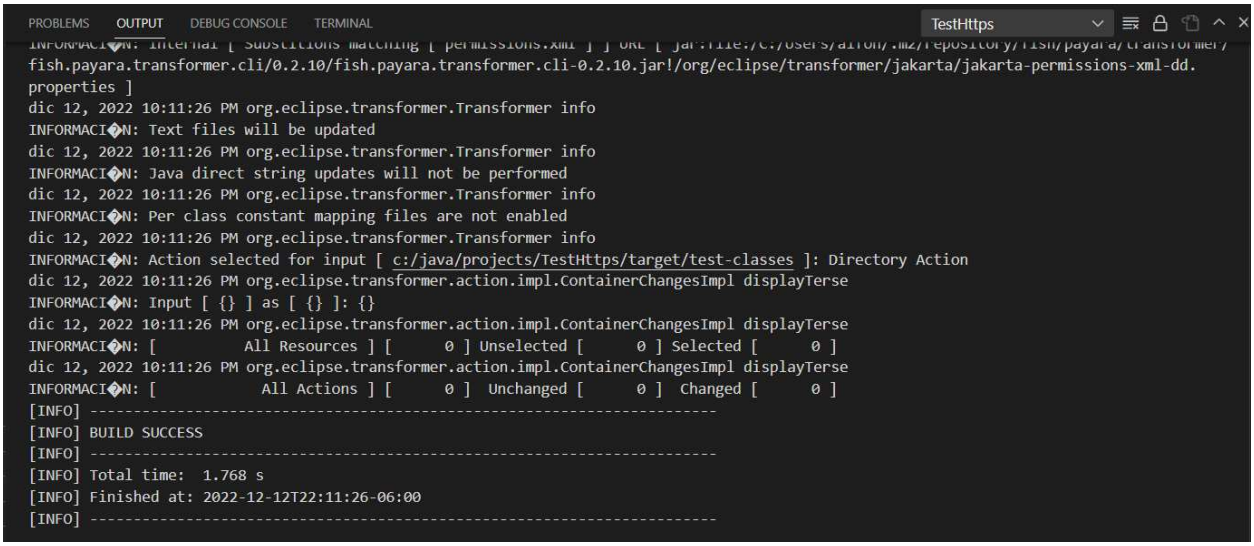
To transform a single file, select the file, right-click on it and select the **Transform to Jakarta EE 10** option:



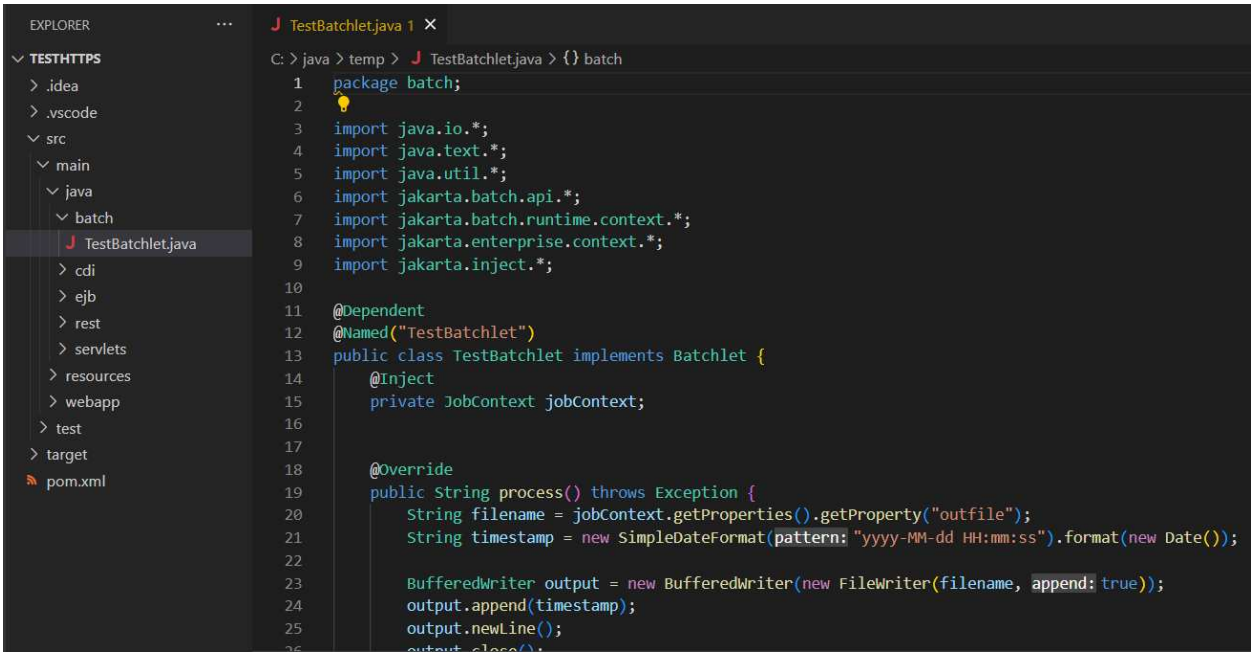
Proceed to select the destination folder of the transformed file:



Finally you can check the output of the transformation process using the **Output** tab:



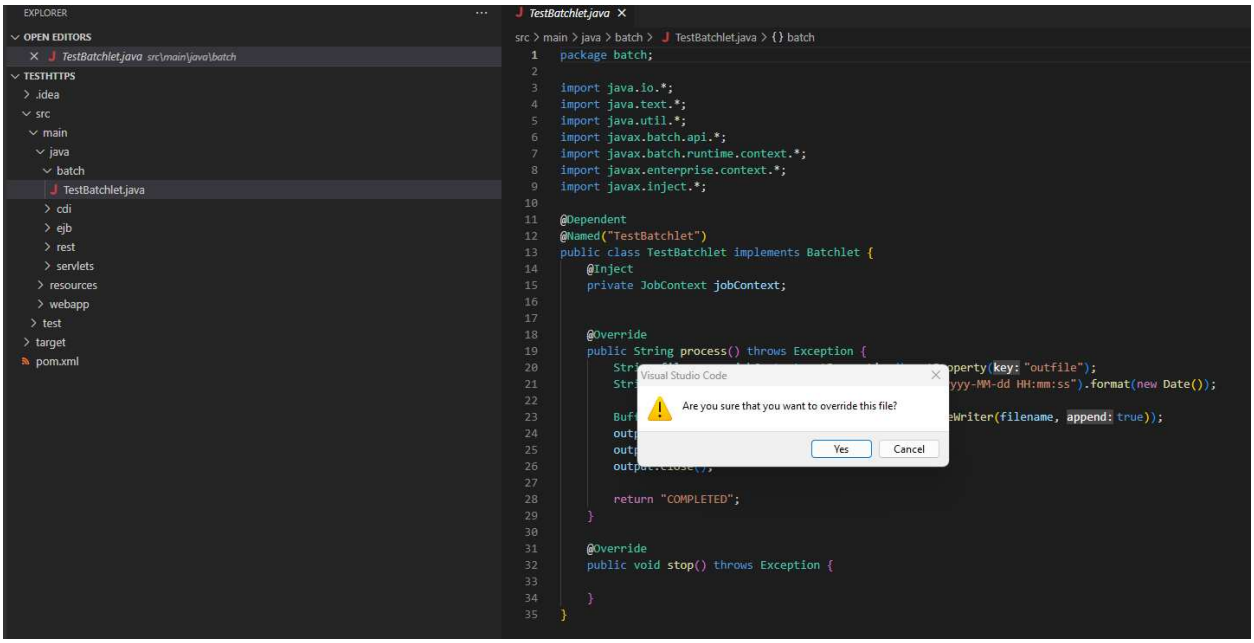
To verify that the resulting file is valid you'll have to open it using either VSCode or a different editor.



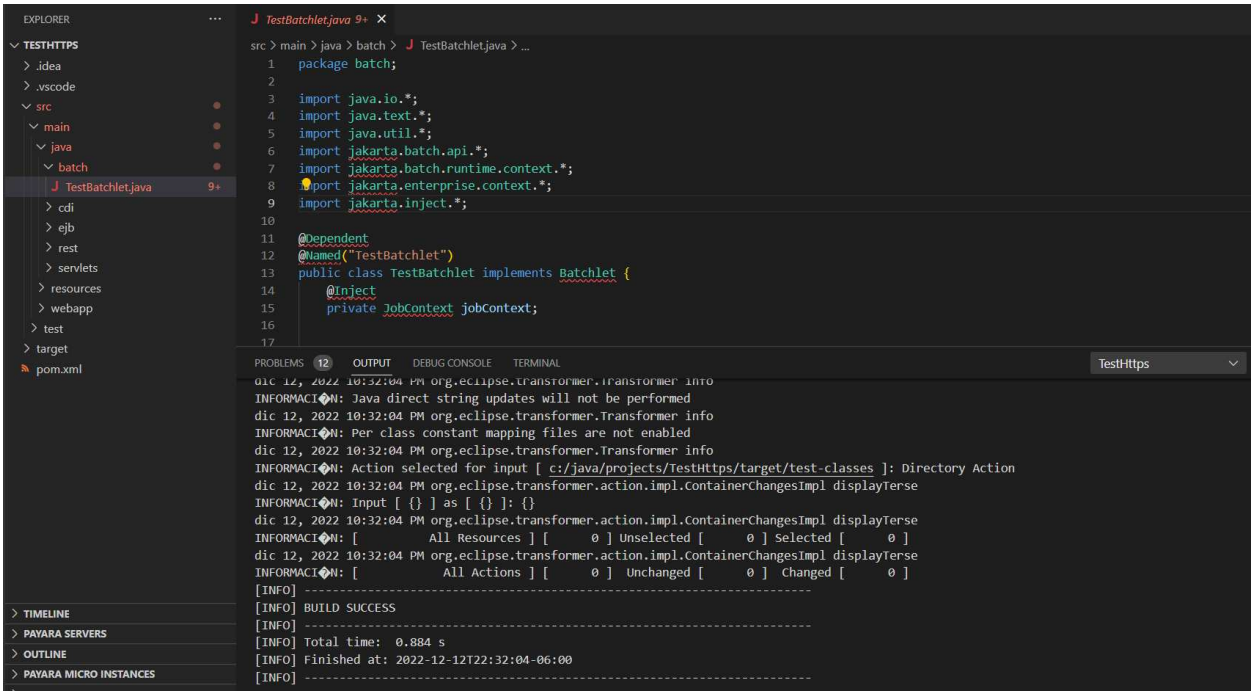
Transform a Single File in the Same Location

It is possible to save the transformed file in the same location where the original resides.

To do this, select the same location folder when inputting the destination folder:

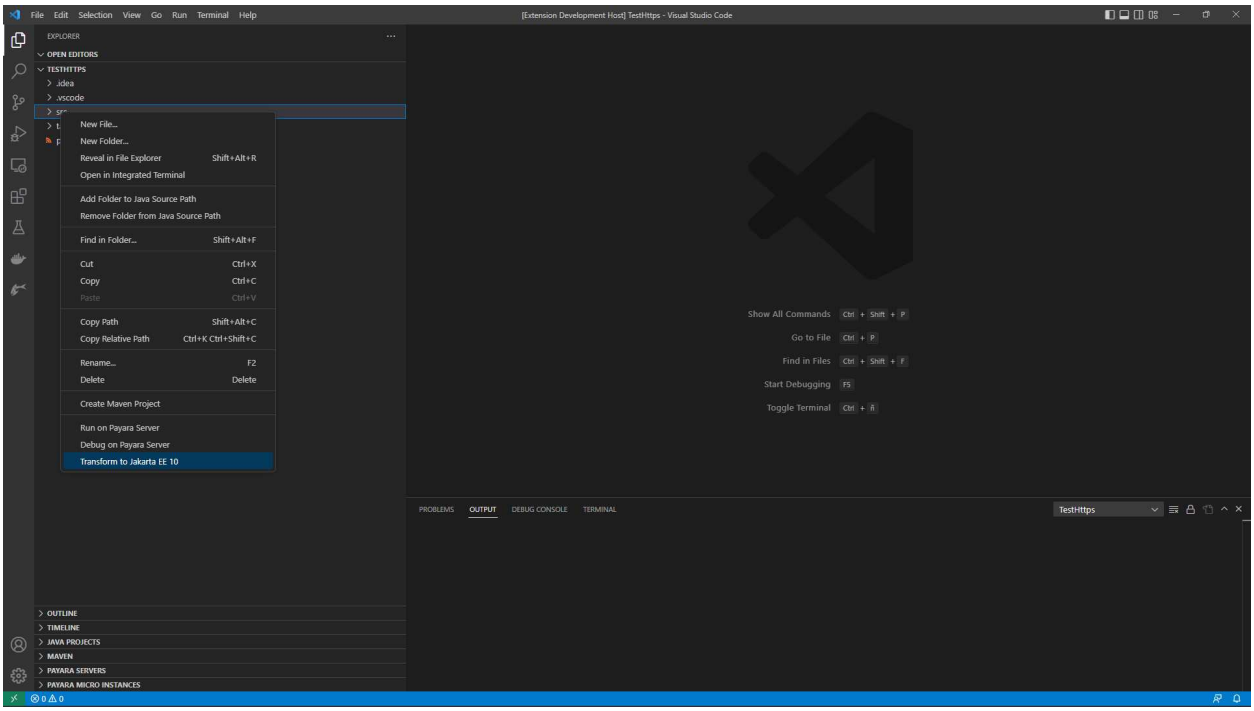


Once the file has been transformed, it will override the current file in the IDE:

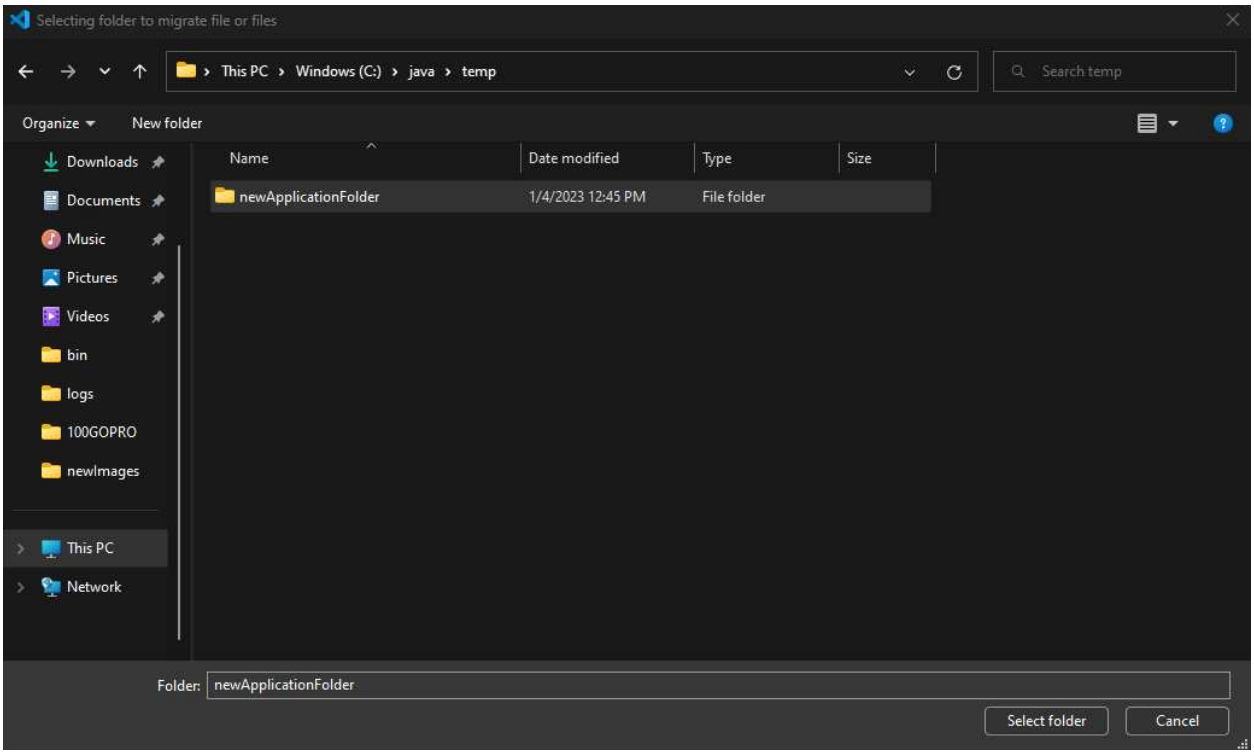


Transform All Files in a Folder

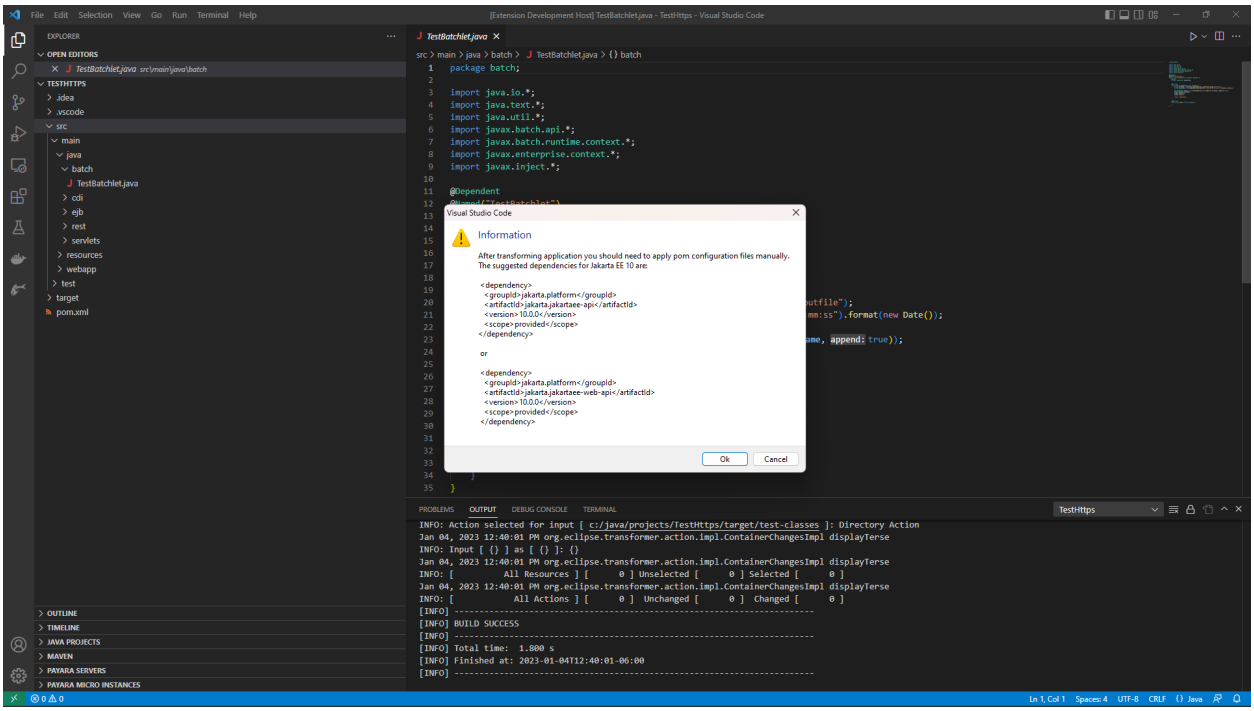
To transform all source code files located in a folder, select the folder, right click and select the **Transform to Jakarta EE 10** option.



Proceed to select the destination folder of the transformed files:



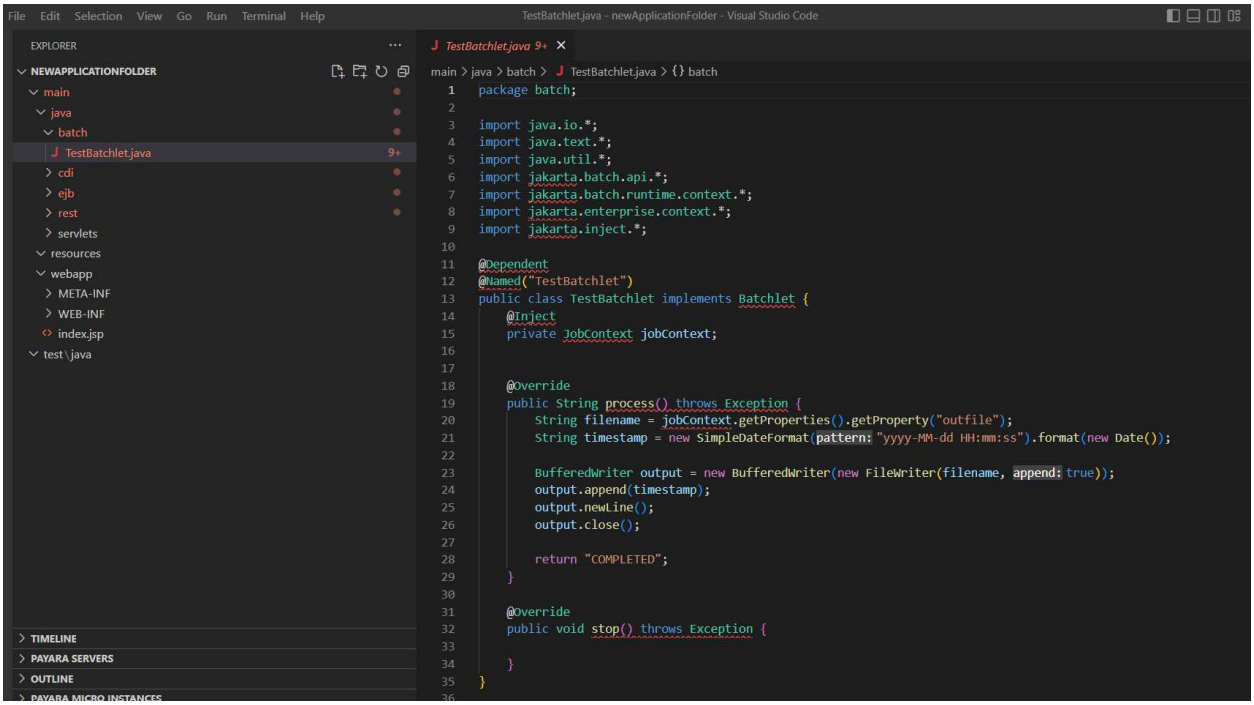
Finally you can check the output of the transformation process using the **Output** tab:



NOTE

The IDE will also show a dialog with additional recommendations to configure the source code's project POM configuration. These configuration changes will have to be done manually if required.

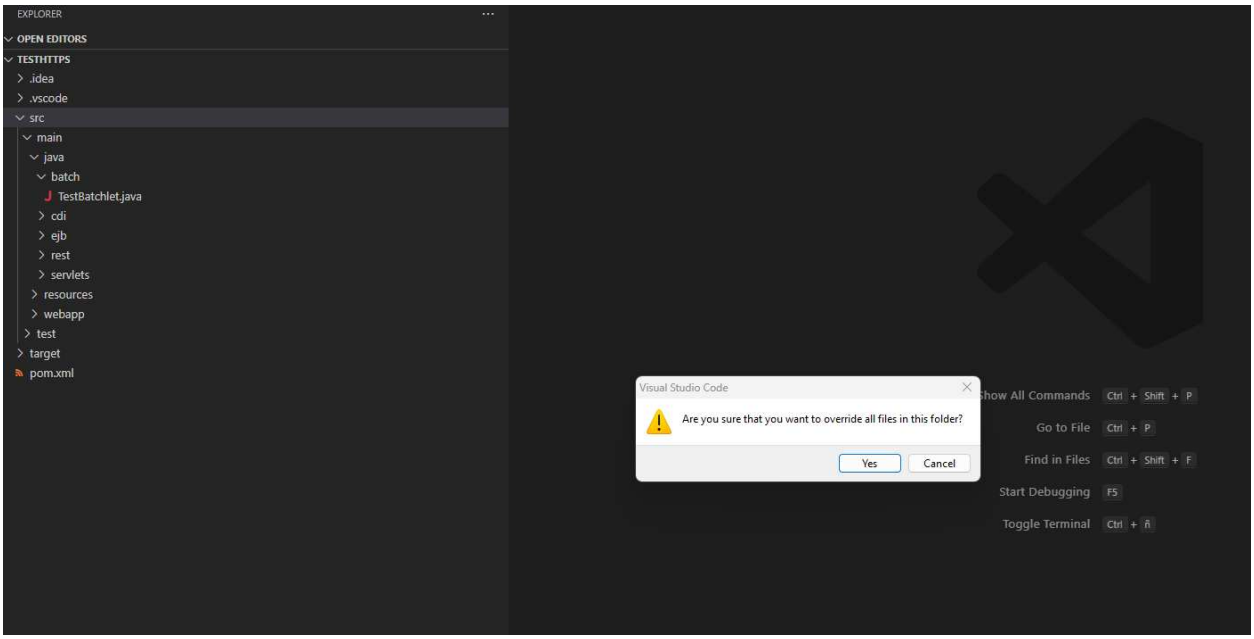
To verify that the resulting files in the new folder are valid, open it from within the IDE.



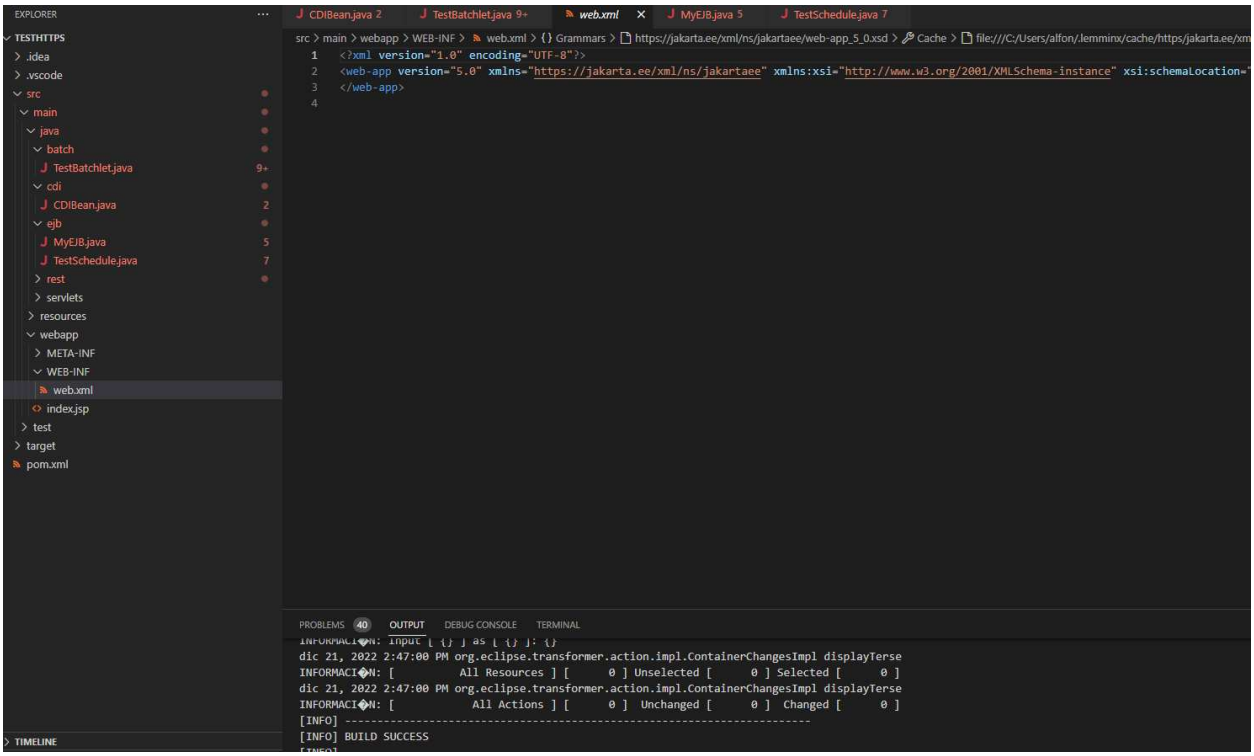
Transform All Files in a Folder in the Same Location

It is possible to save all transformed files in the same location where the originals reside, effectively overriding them.

To do this, select the same location folder when inputting the destination folder:



Once the contents of the folder have been transformed, the new files will override the current ones in the IDE:



Hot Deploy and Auto Deploy

IMPORTANT	To enable these features, the relevant ecosystem plugin to your IDE of choice must be installed.

The Azul Payara IDE ecosystem integrations fully support Hot Deployment and AutoDeployment of applications on a pre-configured Payara Server installation. Once enabled on your IDE of choice, you can simply deploy web applications to Payara Server (and Micro) and modify the source code of any Java class that belong to the application in question, upon save, any changed files within the application will be redeployed.

CAUTION	Both features are not available yet in the IntelliJ IDEA ecosystem integrations.

Hot Deployment

Azul Payara provides the **Hot Deployment** capability, which boosts the performance of the application redeployment operation and dynamically replaces the classloader of the application on each redeployment when the feature is turned on.

TIP	Hot deploy is significantly faster than a vanilla redeployment, boosting

> developer productivity.

Auto Deployment

Azul Payara also allows for a full redeployment of the application automatically via the **Auto Deployment** feature. It will un-deploy the application (if already deployed) and redeploy the new version of the application on every save action after making changes to the source code on your IDE of choice.

WARNING	As the Auto Deploy feature performs a full redeploy of an application, it is slower than hot redeploy, which redeploys only the relevant updated components of the application in question.

Configuration

To configure the settings of these features on your IDE of choice, follow the instructions on the corresponding sections of each integration's documentation:

- Eclipse IDE:
  - hot-deploy-and-auto-deploy-settings
  - hot-deploy-and-auto-deploy-settings
- Apache Netbeans:
  - hot-deploy-and-auto-deploy-settings
  - hot-deploy-and-auto-deploy-settings
- Visual Studio Code:
  - hot-deploy-and-auto-deploy-settings
  - hot-deploy-and-auto-deploy-settings

Connector Suites

Arquillian Containers

Arquillian Container Adapters

An **Arquillian container**, generally speaking, is a runtime with which Arquillian interacts

when running an integration test. Arquillian may manage the container's lifecycle or simply bind to it. Arquillian may deploy archives or descriptors to the container. Arquillian may transfer execution of the tests into the container, or it may simply make the URL of the deployed application available to the test which is acting as a client.

Arquillian supports multiple containers (among them Azul Payara distributions) through its extension mechanism. A **container adapter** is packaged as a JAR and is referenced, along with its dependencies, through as set of Maven GAV coordinates (`groupId`, `artifactId`, `version`). During the execution of tests, a container adapter must be present in the test classpath, allowing Arquillian to locate, load and use it.

In this context, the following container adapters have been implemented within the Payara Ecosystem tools to provide integration with Azul Payara distributions containers:

- For Payara Server:
  - "Payara Server Embedded (`server-embedded`)"
  - "Payara Server Managed (`server-managed`)"
  - "Payara Server Remote (`server-remote`)"
- For Payara Micro:
  - "Payara Micro Managed (`micro-managed`)"

Where Can I Get the Adapters?

The Payara Arquillian container adapters are freely available in Maven Central under the following GAV coordinates :

```
<dependency>
    <groupId>fish.payara.arquillian</groupId>
    <artifactId>arquillian-payara-${container-name}</artifactId>
    <version>${release-id}</version>
</dependency>
```

Where `${container-name}` refers to the specific container in question (see their names in the previous section) and the `${version}` corresponds to the specific adapter version compatible with this Azul Payara version.

TIP | All adapters share the same version as part of their connector suite.

NOTE | For Azul Payara 7.x, adapters in version `4.x` are compatible.

General Usage

An example application demonstrating how to create a test suite that runs the tests inside various Azul Payara distributions using Arquillian can be seen in the Payara Examples repository.

See specifically the ecosystem/arquillian-example subproject for an example of writing integration tests using Arquillian along with the corresponding container adapters described above.

Arquillian requires Junit4, but also offers a version compatible with Junit5. The Payara Arquillian connector remains compatible with this version of Arquillian. See ecosystem/arquillian-junit5-example subproject for the same integration test mentioned above, adapted to work with Junit5.

Payara Server Embedded Arquillian Container Adapter

When using this adapter, the Payara Server container lifecycle is managed by Arquillian and resides in the same JVM where tests are executed using a local communication protocol.

Usage

The Payara Server Embedded Arquillian container adapter can be found on Maven Central, and can be included in your project using the following Maven GAV coordinates:

```
<dependency>
  <groupId>fish.payara.arquillian</groupId>
  <artifactId>arquillian-payara-server-embedded</artifactId>
  <version>${release-id}</version>
</dependency>
```

Container Configuration

The Payara Server container can be configured via the `arquillian.xml` file using the

standard Arquillian Container Configuration mechanism.

Table 213. Configuration Options

Name	Description	Default
`bindHttpPort`	The port number of the http-listener for the embedded Payara Server.	`8181`
`bindHttpsPort`	The port number of the https-listener for the embedded Payara Server.	`8182`
`cleanup`	Specifies whether Arquillian should clean up on shutdown. This recursively deletes files in the `instanceRoot` directory.	`true`
`configurationReadOnly`	Specifies whether Payara Server should write back any changes to specified configuration file or `config/domain.xml` at the specified instance root.	`true`
`configurationXml`	Set the location of configuration file i.e. `domain.xml` using which the Payara Server should run.	
`installRoot`	The installation root directory is the parent directory of a Payara Server instance directory and corresponds to the base directory for an installation of Payara Server.	
`instanceRoot`	The instance root directory is the domain directory and embedded Payara Server uses the server instance directory for domain configuration files.	The default domain directory
`resourcesXml`	A comma-separated list of Payara resources.xml files containing resources that will be added to the Payara instance using the `add-resources` command.	
`hazelcastConfigurationFile`	Full path to a hazelcast configuration file used to override the hazelcast bootstrap default configuration.	

Examples

To configure the Arquillian Container options, you need to use an `arquillian.xml` file placed on the test classpath.

Here's an example `arquillian.xml` file. It configures `adminPort` with a static value. It configures `bindHttpsPort` as a value of a system property `my.httpsport`, which

you can specify for example in the maven `surefire` plugin using the `systemPropertyVariables` option or on the command line like `mvn -Dmy.httpsport=8081 test`.

It also overrides hazelcast bootstrap configuration indicating a path for an XML file in property `hazelcastConfigurationFile`.

Example `arquillian.xml` file

```
<?xml version="1.0"?>
<arquillian xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="http://jboss.org/schema/arquillian"
            xsi:schemaLocation="http://jboss.org/schema/arquillian
                http://jboss.org/schema/arquillian/arquillian_1_0.xsd">
    <container qualifier="payara-embedded" default="true">
        <configuration>
            <property name="bindHttpPort">8080</property>
            <property name="bindHttpsPort">${my.httpsport}</property>
            <property name="hazelcastConfigurationFile">/home/payara/hazelcast-
config.xml</property>
        </configuration>
    </container>
</arquillian>
```

If you want to configure more containers, you can switch between them by setting the `arquillian.launch` system property to the container's qualifier. This is how you can do it with the maven `surefire` plugin (`my.httpsport` system property is used to set the `bindHttpsPort` property in `arquillian.xml`):

Example - Surefire plugin configuration in the project's POM file

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
        <systemPropertyVariables>
            <arquillian.launch>payara</arquillian.launch>
            <my.httpsport>8081</my.httpsport>
        </systemPropertyVariables>
    </configuration>
</plugin>
```

Payara Server Managed Arquillian Container Adapter

When using this adapter, the Payara Server container lifecycle is managed by Arquillian by starting a separate JVM process to run Payara Server locally.

NOTE	As a requirement, the adapter will need to use a local installation of Payara Server.

Usage

The Payara Server Managed Arquillian container adapter can be found on Maven Central, and can be included in your project using the following Maven GAV coordinates:

```
<dependency>
  <groupId>fish.payara.arquillian</groupId>
  <artifactId>arquillian-payara-server-managed</artifactId>
  <version>${release-id}</version>
</dependency>
```

Container Configuration

The Payara Server container can be configured via the `arquillian.xml` file using the standard Arquillian Container Configuration mechanism.

The following configuration options are mandatory:

- `payaraHome` - The local Payara Server installation directory. Can be also set using the `payara.home` system property.

The following Arquillian Container configuration options are available:

Table 214. Configuration Options

Name	Description	Default
`adminHost`	The host to be used to access the Payara Server Admin interface.	`localhost`
`adminHttps`	You can use it to specify whether the HTTP or HTTPS protocol shall be used to access the DAS. The property value can be `true` or `false`.	`false`
`adminPort`	The port to be used to access the Payara Server Admin interface.	`4848`
`adminUser`	The name of the admin user of your DAS. If omitted, no authentication will be used to access the DAS. In this case the `adminPassword` property should also be empty.	

Name	Description	Default
`adminPassword`	The password of the admin user of your DAS. Cannot be omitted if you declare the admin user. If no authentication will be used to access the DAS you must have empty password for your admin user of your domain.	
`allowConnectingToRunningServer`	Allow Arquillian to use an already running Payara Server instance and skips `start-domain` command invocation.	`false`
`authorisation`	If true, basic access authentication is enabled. It also indicates that remote server requires an `adminUser` and `adminPassword` configuration properties.	`false`
`debug`	Starts the server in debug mode using standard Payara Server debug port.	`false`
`domain`	The Payara Server domain to use or the default domain if not specified.	`domain1`
`enableH2`	Starts/Stops the registered H2 database server using the standard H2 port.	`false`
`ignoreCertificates`	If set to `true`, SSL certificate correctness is ignored. This is useful for Docker images / Testcontainers when certificates are not easily matched to internally generated ones and host names are also very hard to match.	`false`
`libraries`	A comma-separated list of library JAR files. Specify the library JAR files by their relative or absolute paths. Specify relative paths relative to `domain-dir/lib/applibs`. The libraries are made available to the application in the order specified.	
`outputToConsole`	Show the output of the admin commands on the console.	`true`

Name	Description	Default
`payaraHome`	The local Payara Server installation directory. Mandatory	Value of the `payara.home` system property.
`properties`	Optional keyword-value pair that specify additional properties for the deployment. The available properties are determined by the implementation of the component that is being deployed.	

Name	Description	Default
`target`	The deployment target of the applications involved in the test. Valid values of the target are: **`server`**: Deploys the component to the default Admin Server instance (on your DAS server). This is the default value if the property is omitted. **`instance_name`**: Deploys the component to the specified stand-alone server instance, which may be on the same hosts or can be on a different one as the DAS server. **`deployment_group_name`**: Deploys the component to every server instance in the specified deployment group. They can be on the same or on several other hosts as the DAS server. NOTE | Arquillian will use only one instance to run the test case. The domain name as a target is not a reasonable deployment scenario in case of testing. The HOST address and port numbers of the test server instance used by Arquillian (determined by the target property) will be automatically retrieved from the DAS server. TIP | You have to make them accessible for your test environment (consider any firewall or proxy configuration).	`server`
`type`	If set to `osgi`, the component is packaged as an OSGi bundle. If the component is packaged as a regular archive, do not set this option.	

The context root that will be used to run the tests is also retrieved automatically from the DAS server. If the deployed application does have a `sun-web.xml`, `glassfish-web.xml` or `payara-web.xml` deployment descriptor, the container will use the name of your deployment without the extension as context root.

The same rule is applied for enterprise applications if there is no `application.xml` file. The JAR test-deployments are treated as a web application.

IMPORTANT	glassfish-web.xml and sun-web.xml are deprecated and support will be removed in a future release. It is recommended to use payara-web.xml instead.

Examples

To configure the Arquillian Container options, you need to use an `arquillian.xml` file placed on the test classpath.

Here's an example `arquillian.xml` file. It configures `adminPort` with a static value.

It will configure the `adminPassword` as a value of a system property `my.admin.password`, which you can specify for example in the maven `surefire` plugin using the `systemPropertyVariables` option or on the command line like `mvn -Dmy.admin.password=mypassword test`.

Example `arquillian.xml` file

```
<?xml version="1.0"?>
<arquillian xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="http://jboss.org/schema/arquillian"
            xsi:schemaLocation="http://jboss.org/schema/arquillian
                http://jboss.org/schema/arquillian/arquillian_1_0.xsd">
    <container qualifier="payara" default="true">
        <configuration>
            <property name="adminPort">4848</property>
            <property name="adminPassword">${my.admin.password}</property>
        </configuration>
    </container>
</arquillian>
```

If you want to configure more containers, you can switch between them by setting the `arquillian.launch` system property to the container's qualifier.

This is how you can do it with the maven `surefire` plugin (`my.admin.password` system property is used to set the `adminPassword` property in `arquillian.xml`):

Example - Surefire plugin configuration in the project's POM file

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
        <systemPropertyVariables>
            <arquillian.launch>payara</arquillian.launch>
            <payara.home>/path/to/payara</payara.home>
            <my.admin.password>mypassword</my.admin.password>
        </systemPropertyVariables>
    </configuration>
</plugin>
```

Downloading Payara Server Automatically from Maven

You can configure your Maven project to automatically download and install the Payara Server version required by the container adapter.

To set up this installation, you can use the Maven Dependency Plugin (specifically the `unpack` goal) as follows:

Example - Dependency plugin configuration in the project's POM file

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <executions>
            <execution>
                <id>unpack</id>
                <phase>process-test-classes</phase>
                <goals>
                    <goal>unpack</goal>
                </goals>
                <configuration>
                    <artifactItems>
                        <artifactItem>
                            <groupId>fish.payara.distributions</groupId>
                            <artifactId>payara</artifactId>
                            <version>${payara.version}</version>
                            <type>zip</type>
                            <overWrite>false</overWrite>

<outputDirectory>${project.build.directory}</outputDirectory>
                        </artifactItem>
                    </artifactItems>
                </configuration>
            </execution>
        </executions>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
```

```
        <artifactId>maven-surefire-plugin</artifactId>
        <configuration>
            <systemPropertyVariables>
                <payara.home>${project.build.directory}/payara5</payara.home>
            </systemPropertyVariables>
        </configuration>
    </plugin>
</plugins>
```

Payara Server Remote Arquillian Container Adapter

When using this adapter, the Payara Server container lifecycle is NOT managed by Arquillian. Instead, the adapter will connect to an already running Payara Server instance via a remote communication protocol by talking to the server's DAS process directly. In this scenario, the integration test setup will have to bootstrap the server's domain startup and shutdown on its own.

This gives you the flexibility to use any possible deployment scenarios having your servers either on many different physical or virtual nodes, or on the same one.

Usage

The Payara Server Remote Arquillian container adapter can be found on Maven Central, and can be included in your project using the following Maven GAV coordinates:

```
<dependency>
  <groupId>fish.payara.arquillian</groupId>
  <artifactId>arquillian-payara-server-remote</artifactId>
  <version>${release-id}</version>
</dependency>
```

Container Configuration

The Payara Server container can be configured via the `arquillian.xml` file using the standard Arquillian Container Configuration mechanism.

The following Arquillian Container configuration options are available:

Table 215. Configuration Options

Name	Description	Default
`adminHost`	The host to be used to access the Payara Server Admin interface.	`localhost`

Name	Description	Default
`adminHttps`	You can use it to specify whether the HTTP or HTTPS protocol shall be used to access the DAS. The property value can be `true` or `false`.	`false`
`adminPort`	The port to be used to access the Payara Server Admin interface.	`4848`
`adminUser`	The name of the admin user of your DAS. If omitted, no authentication will be used to access the DAS. In this case the `adminPassword` property should also be empty.	
`adminPassword`	The password of the admin user of your DAS. Cannot be omitted if you declare the admin user. If no authentication will be used to access the DAS you must have empty password for your admin user of your domain.	
`authorisation`	If true, basic access authentication is enabled. It also indicates that remote server requires an `adminUser` and `adminPassword` configuration properties.	`false`
`ignoreCertificates`	If set to `true`, SSL certificate correctness is ignored. This is useful for Docker images / Testcontainers when certificates are not easily matched to internally generated ones and host names are also very hard to match.	`false`
`libraries`	A comma-separated list of library JAR files. Specify the library JAR files by their relative or absolute paths. Specify relative paths relative to `domain-dir/lib/applibs`. The libraries are made available to the application in the order specified.	
`properties`	Optional keyword-value pair that specify additional properties for the deployment. The available properties are determined by the implementation of the component that is being deployed.	

Name	Description	Default
`target`	The deployment target of the applications involved in the test. Valid values of the target are: **`server`**: Deploys the component to the default Admin Server instance (on your DAS server). This is the default value if the property is omitted. **`instance_name`**: Deploys the component to the specified stand-alone server instance, which may be on the same hosts or can be on a different one as the DAS server. **`deployment_group_name`**: Deploys the component to every server instance in the specified deployment group. They can be on the same or on several other hosts as the DAS server. The domain name as a target is not a reasonable deployment scenario in case of testing. The HOST address and port numbers of the test server instance used by Arquillian (determined by the target property) will be automatically retrieved from the DAS server. **TIP** | You have to make them accessible for your test environment (consider any firewall or proxy configuration).	`server`
`type`	If set to `osgi`, the component is packaged as an OSGi bundle. If the component is packaged as a regular archive, do not set this option.	

The context root that will be used to run the tests is also retrieved automatically from the DAS server. If the deployed application does have a `sun-web.xml`, `glassfish-web.xml` or `payara-web.xml` deployment descriptor, the container will use the name of your deployment without the extension as context root.
The same rule is applied for enterprise applications if there is no `application.xml` file. The JAR test-deployments are treated as a web application.

IMPORTANT	glassfish-web.xml and sun-web.xml are deprecated and support

> will be removed in a future release. It is recommended to use payara-web.xml instead.

Examples

To configure the Arquillian Container options, you need to use an `arquillian.xml` file placed on the test classpath. Here's an example `arquillian.xml` file. It configures `adminPort` with a static value. It configures `adminPassword` as a value of a system property `my.admin.password`, which you can specify for example in the maven `surefire` plugin using the `systemPropertyVariables` option or on the command line like `mvn -Dmy.admin.password=mypassword test`.

Example `arquillian.xml` file

```
<?xml version="1.0"?>
<arquillian xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="http://jboss.org/schema/arquillian"
            xsi:schemaLocation="http://jboss.org/schema/arquillian
                http://jboss.org/schema/arquillian/arquillian_1_0.xsd">
    <container qualifier="payara" default="true">
        <configuration>
            <property name="adminPort">4848</property>
            <property name="adminPassword">${my.admin.password}</property>
        </configuration>
    </container>
</arquillian>
```

If you want to configure more containers, you can switch between them by setting the `arquillian.launch` system property to the container's qualifier.

This is how you can do it with the maven `surefire` plugin (`my.admin.password` system property is used to set the `adminPassword` property in `arquillian.xml`):

Example - Surefire plugin configuration in the project's POM file

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
        <systemPropertyVariables>
            <arquillian.launch>payara</arquillian.launch>
            <my.admin.password>mypassword</my.admin.password>
        </systemPropertyVariables>
    </configuration>
```

```
</plugin>
```

Payara Micro Managed Arquillian Container Adapter

When using this adapter, the Payara Micro container lifecycle is managed by Arquillian by starting a separate JVM process to launch and stop a Payara Micro instance locally.

Usage

The Payara Micro Managed Arquillian container adapter can be found on Maven Central, and can be included in your project using the following Maven GAV coordinates:

```
<dependency>
  <groupId>fish.payara.arquillian</groupId>
  <artifactId>arquillian-payara-micro-managed</artifactId>
  <version>${release-id}</version>
</dependency>
```

Container Configuration

The Payara Micro container can be configured via the `arquillian.xml` file using the standard Arquillian Container Configuration mechanism.

The following configuration options are available:

Table 216. Configuration Options

Container Option	Description	System Property	Environment Variable	Default
`microJar`	Provides the location of the Payara Micro Jar.	`payara.microJar`	`MICRO_JAR`	
`startupTimeoutInSeconds`	Specifies the amount of time in seconds that the container will wait for Payara Micro to start.	`payara.startupTimeoutInSeconds`	`MICRO_STARTUP_TIMEOUT_IN_SECONDS`	`180`
`randomHttpPort`	Randomises the initial HTTP port of Payara Micro. This saves time by avoiding collisions with the default port.	`payara.randomHttpPort`	`MICRO_RANDOM_HTTP_PORT`	`true`

Container Option	Description	System Property	Environment Variable	Default
`autoBindHttp`	Enables the `--autoBindHttp` option for the Micro instance.	`payara.autoBindHttp`	`MICRO_AUTOBIND_HTTP`	`true`
`clusterEnabled`	Enables clustering on the Micro instance.	`payara.clusterEnabled`	`MICRO_CLUSTER_ENABLED`	`false`
`consoleOutput`	Enables/disables console output on the Micro instance.	`payara.consoleOutput`	`MICRO_CONSOLE_OUTPUT`	`true`
`debug`	Enables debugging on the Payara Micro instance. By default, Payara Micro will wait for the debugger on port `5005` before starting. Disables the startup timeout. To change this, provide java debug options to `cmdOptions`.	`payara.debug`	`MICRO_DEBUG`	`false`
`cmdOptions`	Provides JVM options to the Java process running the Micro instance.	`payara.cmdOptions`	`MICRO_CMD_OPTIONS`	
`extraMicroOptions`	Provides standard Payara Micro arguments for the server instance to launch with	`payara.extraMicroOptions`	`EXTRA_MICRO_OPTIONS`	

Examples

To configure the options using Arquillian container properties, you need to use an `arquillian.xml` file placed on the test classpath.

Here's an example `arquillian.xml` file. It configures `randomHttpPort` and `autoBindHttp` options.

Example `arquillian.xml` file

```
<?xml version="1.0"?>
<arquillian xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="http://jboss.org/schema/arquillian"
            xsi:schemaLocation="http://jboss.org/schema/arquillian
                http://jboss.org/schema/arquillian/arquillian_1_0.xsd">
    <container qualifier="payara-micro-managed" default="true">
        <configuration>
            <property name="randomHttpPort">false</property>
            <property name="autoBindHttp">false</property>
        </configuration>
    </container>
</arquillian>
```

You can configure the options also using system properties. Here's an example configuration if you run your tests using the maven `surefire` plugin:

Example - Surefire plugin configuration in the project's POM file

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
        <systemPropertyVariables>
            <arquillian.launch>payara-micro-managed</arquillian.launch>
            <payara.randomHttpPort>false</payara.randomHttpPort>
            <payara.autoBindHttp>false</payara.autoBindHttp>
        </systemPropertyVariables>
    </configuration>
</plugin>
```

You can also configure the options using environment variables, for example when running the maven `mvn test` command:

```
export MICRO_RANDOM_HTTP_PORT=false
export MICRO_AUTOBIND_HTTP=false

mvn test
```

Cloud Connectors

Cloud Connectors

The Payara Cloud Connectors are a collection of JCA adapters for use with various popular cloud messaging providers. These connectors can be deployed to either Payara Server or Payara Micro.

The following connectors are available:

- "Apache Kafka"
- "Amazon SQS"
- "MQTT"
- "Azure Service Bus"

Obtaining the Connectors

Nexus

Cloud Connectors are available in Binary form to Enterprise License users in the `payara-enterprise-downloadable-artifacts` repository.

The artifacts can be found in `/fish/payara/cloud/connectors`.

Building from Source

Cloud Connectors can be built from source as follows:

Clone the Cloud-Connectors git repository:

```
git clone git@github.com:payara/Cloud-Connectors.git
```

Build and install the connectors:

```
cd Cloud-Connectors/
mvn clean install
```

The connectors can be found in the following locations as JCA RARs:

- Kafka - ./Kafka/KafkaRAR/target/kafka-rar-0.8.0.rar
- Amazon SQS - ./AmazonSQS/AmazonSQSRAR/target/amazon-sqs-rar-2.1.0.rar
- MQTT - ./MQTT/MQTTRAR/target/mqtt-rar-0.8.0.rar
- Azure Service Bus - ./AzureServiceBus/AzureSBRAR/target/azure-sb-rar-0.8.0.rar

Installing a Connector

Installing Cloud Connectors for Payara Server and Payara Micro both require the Cloud Connector jar to be deployed. However, Payara Server requires some additional configuration to classloading.

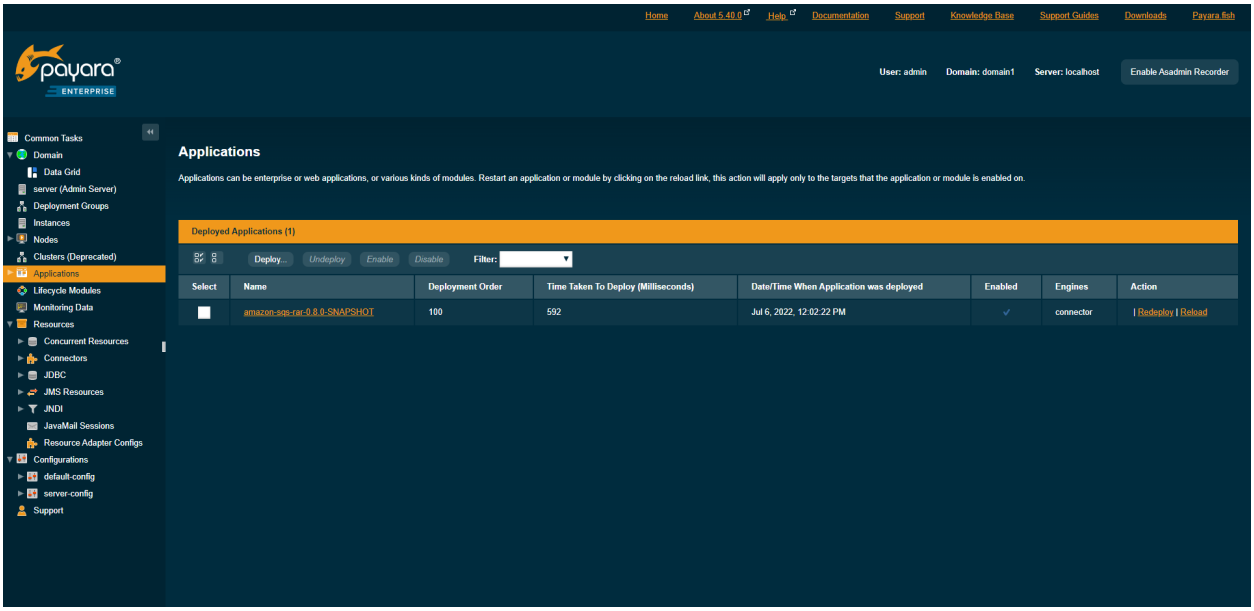
A connector (JCA rar) is installed by deploying it in the same way as an application archive (e.g. a .war) is deployed:

Payara Server

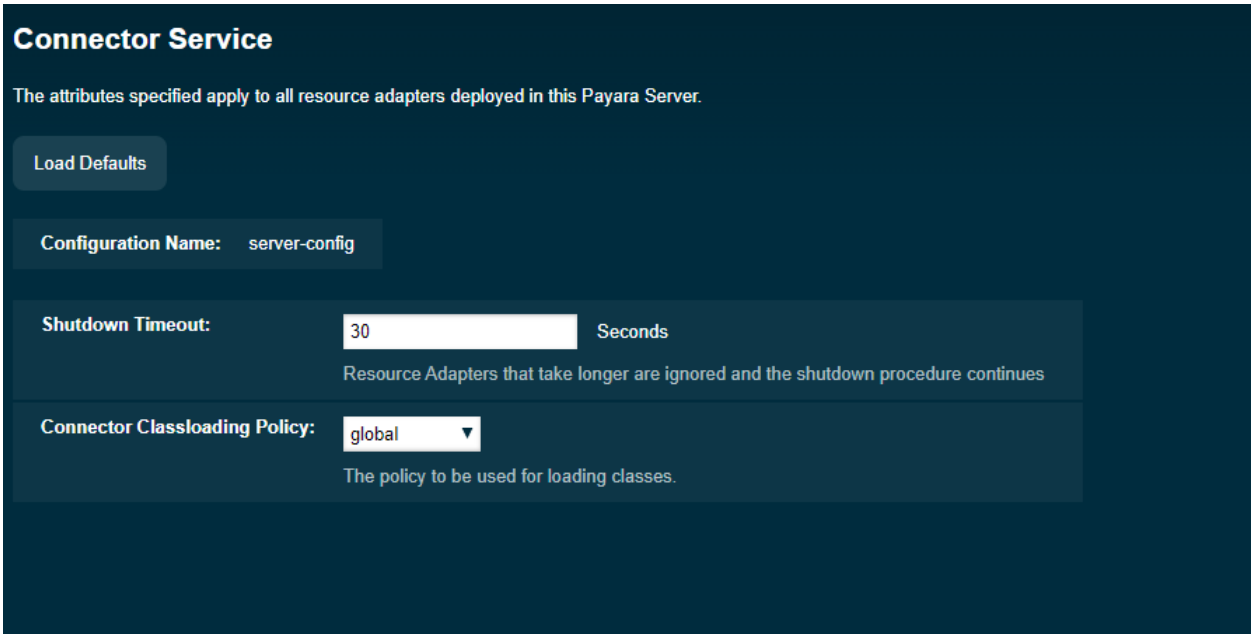
You can configure Payara Server through the Admin Console or through the Asadmin CLI.

Using the Admin Console

You can deploy the connector with the admin console;



Then configure the `Connector Classloading Policy` to `Global` in `{config} > Connector Service > Connector Classloading Policy`.



This allows the classes from the cloud connector to be globally available in the server.

Using the Asadmin CLI

Deploy the application with the CLI;

```
asadmin> deploy --target=server amazon-sqs-rar.rar
```

Configure the classloading policy;

```
asadmin> set configs.config.server-config.connector-service.class-loading-
policy=global
```

Payara Micro

```
java -jar payara-micro.jar --autobindhttp --deploy kafka-rar-0.8.0.rar
```

The connector is subsequently available to all other applications deployed to Payara Micro. An application using a connector and the connector itself can be deployed simultaneously:

```
java -jar payara-micro.jar --autobindhttp --deploy kafka-rar-0.8.0.rar --deploy
myapp.war
```

Securing Credentials

In the documentation pages for the currently available cloud connectors, much of the configuration shown is done via annotations. This configuration contains credentials as well, which as a rule of thumb should not directly be placed in code. On Payara all properties can be replaced via System properties using the syntax ${system.property.name} or environment variables using the syntax `${ENV=evironment.property.name}` or password aliases using the syntax `${ALIAS=alias.name}`.

Amazon SQS

Amazon SQS Cloud Connector

Amazon SQS (Simple Queue Service) is a hosted and highly scalable distributed message queue service.

To get the correct version of the artifact to use in your application (since it depends on a specific version of the Amazon Web Services Java SDK), please follow the instructions detailed in "Versioning".

Once deployed, a message similar to this one should be printed out to the server's output:

```
[fish.payara.cloud.connectors.amazonsqs.api.inbound.AmazonSQSResourceAdapter] [tid:
_ThreadID=1 _ThreadName=main] [timeMillis: 1495398495490] [levelValue: 800] Amazon
SQS Resource Adapter Started..
```

Sending Messages

Sending messages to Amazon SQS can be done via the JCA and an Amazon specific API. In order to start using this API to send messages, a resource has to be defined via the JCA API; a connection factory.

The connection factory has to be given a name, which can be any name that is valid for JNDI. The `java:app` namespace is typically recommended to be used. The type of the connection factory to be used for Amazon SQS is `fish.payara.cloud.connectors.amazonsqs.api.AmazonSQSConnectionFactory`, and we have to specify the resource adapter name which is here `amazon-sqs-`

`rar-2.1.0.rar`.

Finally, the AWS account details have to be specified in the properties of the corresponding JMS resource definitions.

The following gives an example:

```
@ConnectionFactoryDefinition (
  name = "java:app/amazonsqs/factory",
  interfaceName =
"fish.payara.cloud.connectors.amazonsqs.api.AmazonSQSConnectionFactory",
  resourceAdapter = "amazon-sqs-rar-2.1.0",
  properties = {"awsAccessKeyId=<accessKeyID>", "awsSecretKey=<secretKey>",
"region=eu-west-2"}
)
public class SendSQSMessage {
}
```

With the above definition in place, the following code shows an example of sending a message:

```
@Singleton
@Startup
public class SendSQSMessage {

 @Resource(lookup = "java:app/amazonsqs/factory")
 private AmazonSQSConnectionFactory factory;

 @PostConstruct
 public void init() {
    try (AmazonSQSConnection connection = factory.createConnection()) {
        SendMessageRequest sendMsgRequest = SendMessageRequest.builder()
                        .queueUrl("<queueURL>")
                        .messageBody("Hello World")
                        .build();
        connection.sendMessage(sendMsgRequest);
    }
    catch (Exception ex) {
    }
 }
}
```

Receiving Messages

Messages can be received from Amazon SQS by creating an MDB (Message Driven Bean) that implements the

`fish.payara.cloud.connectors.amazonsqs.api.AmazonSQSListener`

marker interface and has a single method annotated with `@OnSQSMessage` and the method signature `void method(Message message)`.

The following gives an example:

```
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "awsAccessKeyId", propertyValue =
"someKey"),
    @ActivationConfigProperty(propertyName = "awsSecretKey", propertyValue =
"someSecretKey"),
    @ActivationConfigProperty(propertyName = "queueURL", propertyValue =
"someQueueURL"),
    @ActivationConfigProperty(propertyName = "pollInterval", propertyValue = "1"),
    @ActivationConfigProperty(propertyName = "region", propertyValue = "eu-west-2")
})
public class ReceiveSQSMessage implements AmazonSQSListener {

    @OnSQSMessage
    public void receiveMessage(Message message) {
     // Handle message
    }
}
```

Large Message Support

By default, the AWS Java SDK has a message size limitation of `256KB`, which can be restrictive for certain use cases.

To address this limitation, the Payara Cloud Connector offers enhanced support for sending and receiving larger messages, surpassing the `256KB` restriction by leveraging Amazon S3 storage. To enable this feature, configure the S3 bucket name using the `s3BucketName` property.

Prerequisites for Larger Message Support

Before testing the larger message support feature, ensure the following prerequisites are met:

1. Create an S3 Bucket in AWS:
    - Create a new Amazon S3 bucket in your AWS account where larger messages will be stored.
2. IAM Role Permissions:

- Add `AmazonS3FullAccess` or related permissions to the IAM role associated with your application.
- This permission ensures that your application has the necessary access to the S3 bucket for storing and retrieving larger messages.

Sending Large Messages

To send messages larger than `256KB`, include the `s3BucketName` property in the connection factory definition. Specify the desired AWS S3 bucket name to store these larger messages.

For example:

```
@ConnectionFactoryDefinition (
    name = "java:app/amazonsqs/factory",
    interfaceName =
"fish.payara.cloud.connectors.amazonsqs.api.AmazonSQSConnectionFactory",
    resourceAdapter = "amazon-sqs-rar-2.1.0",
    properties = {"awsAccessKeyId=<accessKeyID>", "awsSecretKey=<secretKey>",
"region=eu-west-2", "s3BucketName=MyS3Bucket"}
)
@Singleton
@Startup
public class SendSQSMessage {
}
```

After configuring the connection factory with the S3 bucket name, you can send larger messages using the provided Amazon SQS API, as demonstrated in the previous examples.

Receiving Large Messages

To receive larger messages, update the Message Driven Bean (MDB) configuration with the `s3BucketName` property:

```
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "awsAccessKeyId", propertyValue =
"someKey"),
    @ActivationConfigProperty(propertyName = "awsSecretKey", propertyValue =
"someSecretKey"),
    @ActivationConfigProperty(propertyName = "queueURL", propertyValue =
"someQueueURL"),
    @ActivationConfigProperty(propertyName = "pollInterval", propertyValue = "1"),
    @ActivationConfigProperty(propertyName = "region", propertyValue = "eu-west-
2"),
```

```
    @ActivationConfigProperty(propertyName = "s3BucketName", propertyValue =
"MyS3Bucket"),
    @ActivationConfigProperty(propertyName = "s3FetchMessage", propertyValue =
"true")
})
public class ReceiveSQSMessage implements AmazonSQSListener {

    @OnSQSMessage
    public void receiveMessage(Message message) {
     // Handle larger message
    }
}
```

Configure the `s3BucketName` property along with other properties to specify the AWS S3 bucket where larger messages are stored.

Configuration Properties

The full list of configuration properties is given below:

Config Property Name	Type	Default	Notes
`awsAccessKeyId`	String	None	Must be set to the access key of your AWS account
`awsSecretKey`	String	None	Must be set to the secret key of your AWS account
`queueURL`	String	None	Must be set to the URL for an SQS queue
`region`	String	None	Must be set to the AWS region name of your queue
`maxMessages`	Integer	10	The maximum number of messages to download on a poll
`initialPollDelay`	Integer	1	The delay (in seconds) before polling the queue after MDB activation (MDB only)
`pollInterval`	Integer	3	How often should the adapter poll for messages (in seconds) (MDB Only)
`messageAttributeNames`	String	All	The list of message attribute names that should be fetched with the message (MDB Only)
`attributeNames`	String	All	The list of attribute names that should be fetched with the message (MDB Only)
`s3BucketName`	String	None	AWS S3 bucket name for storing larger messages

Config Property Name	Type	Default	Notes
`s3SizeThreshold`	Integer	0	AWS S3 size threshold for storing messages (in bytes)
`s3KeyPrefix`	String	None	AWS S3 key prefix for organizing stored messages
`s3FetchMessage`	Boolean	true	Flag to fetch the full message from S3 when it exceeds the size threshold. If set to `false`, only the message metadata for larger messages, including the bucket name and message key, will be returned instead of the full message.

Amazon SQS Versioning

The SQS connector's behavior and features are affected by the exact version of the Amazon Web Services Java SDK version being planned to use. For each version, there are separate versions of the connector's JCA adapter and its API.

The tags below are for the Cloud Connectors repository. You should use the one corresponding to the SQS and Jakarta version you require.

- 0.8.0: Amazon SQS Version 1 using JakartaEE 8 and the `javax` namespace
- AmazonSQS-1.1.0: Amazon SQS Version 2 using JakartaEE 8 and the `javax` namespace
- AmazonSQS-2.1.0: Amazon SQS Version 2 using JakartaEE 10 and the `jakarta` namespace

Amazon SQS Connector Version 1

This is the first version of the connector that is based on Amazon Web Services Java SDK v1.

Usage

The `amazon-sqs-rar-0.8.0.rar` JCA adapter has to be deployed as shown in the Installing-a-connector of the Cloud Connectors overview.

Additionally, to use the connector API add the following Maven dependencies to your

application's project:

```
<dependency>
  <groupId>fish.payara.cloud.connectors.amazonsqs</groupId>
  <artifactId>amazon-sqs-jca-api</artifactId>
  <version>[,1.0.0)</version>
  <type>jar</type>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-java-sdk-sqs</artifactId>
  <version>[1.11.118,)</version>
  <type>jar</type>
  <scope>provided</scope>
</dependency>
```

NOTE

These dependencies have a provided `scope` since the types within this dependency are globally available to every application deployed to Payara Micro after the `amazon-sqs-rar-0.8.0.rar` has been deployed.

Amazon SQS Connector Version 2

This is the second version of the connector that is based on Amazon Web Services SDK v2.

IMPORTANT

If you have selected to use SQS Connector 2.1.0, replace all occurunces of `1.0.0` with `2.1.0` for the following example.

Usage

The `amazon-sqs-rar-2.1.0.rar` JCA adapter can be downloaded from Maven Central, and has to be deployed as shown in the Installing-a-connector of the Cloud Connectors overview.

Additionally, to use the connector API add the following Maven dependencies to your application's project:

```
<dependency>
    <groupId>fish.payara.cloud.connectors.amazonsqs</groupId>
    <artifactId>amazon-sqs-jca-api</artifactId>
```

```
    <version>2.1.0</version>
    <type>jar</type>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>sqs</artifactId>
    <version>[2.25.6,)</version>
    <type>jar</type>
    <scope>provided</scope>
</dependency>
```

NOTE	The `amazon-sqs-jca-api` artifact has to be installed to your local Maven repository first.

Amazon Web Services SSO Integration

If you are using version 2 of the Amazon Web Services Java SDK, then it is possible to configure the SQS connector use Amazon SSO authentication instead of a `secretKey` and `keyID` set of credentials.

The Amazon Single Sign-On (AWS SSO) is also known as the AWS IAM Identity Center. This center is used by AWS administrators to centrally manage the data and access permissions of all user identities. To view more information about it please read here: AWS IAM Identity Center

In this case, instead of using a traditional "Service Account" to allow a deployed application to connect to a SQS Queue, instead you can allow individual AWS accounts to log in via SSO first and then make the calls as long as they have been configured with the right permissions. To configure this you must follow these steps:

1. Configure AWS IAM Identity Center Accounts
2. Configure SQS Queue
3. Download and install the Amazon SQS Connector v2
4. Install the Amazon CLI locally
5. Configure the user credentials

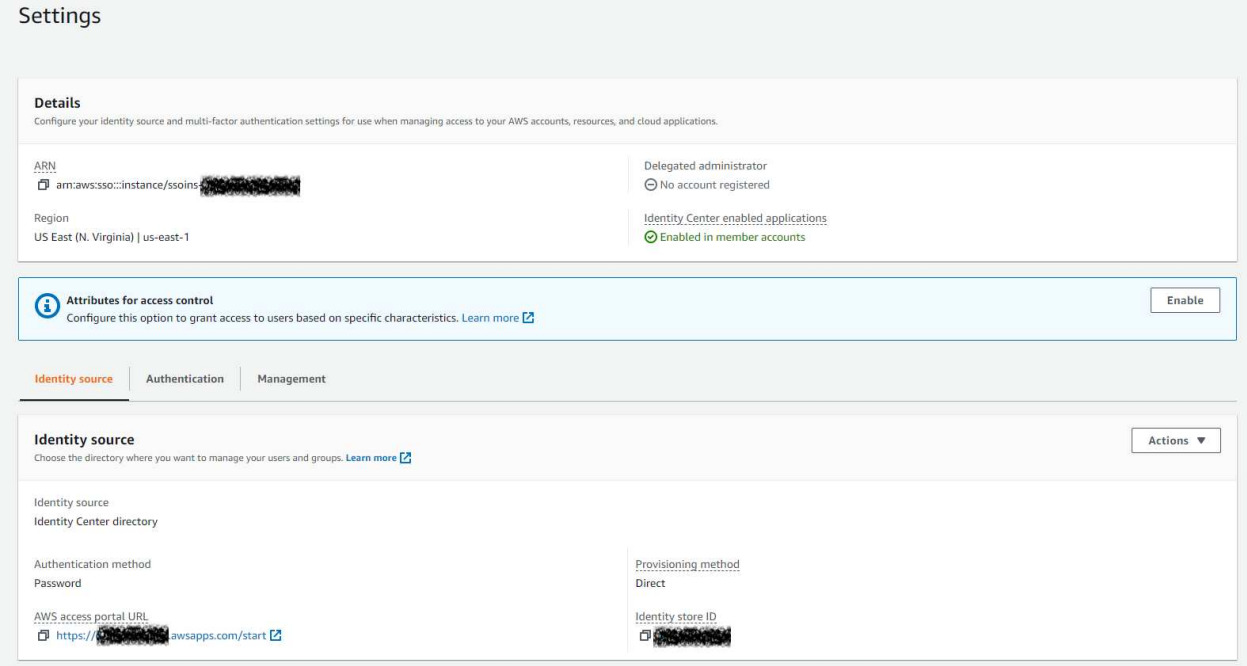
6. Execute the SSO login

IMPORTANT	Using AWS SSO credentials is a recommended practice for development environment only. Please refrain from using this setup on production environments.

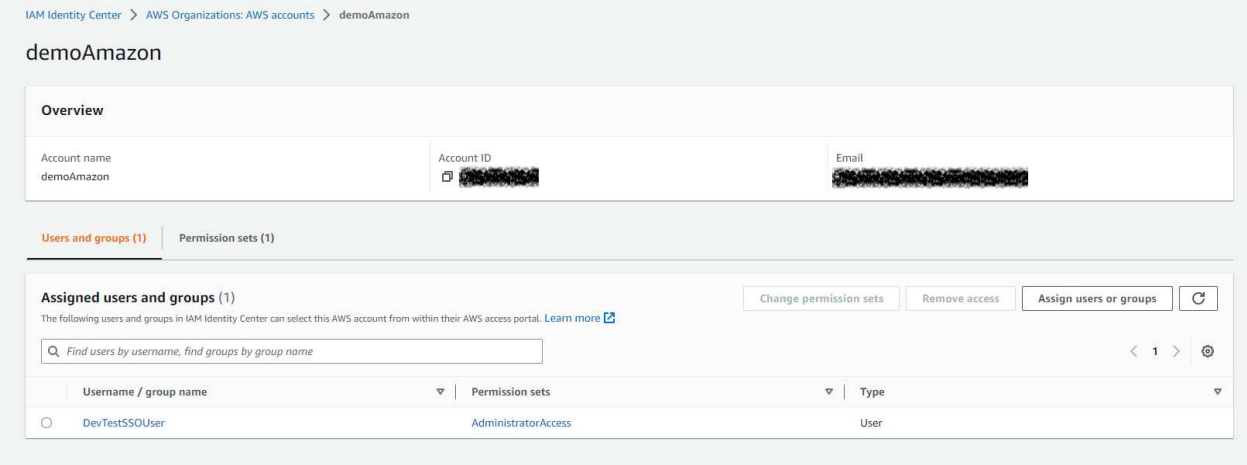
Configure AWS IAM Identity Center Accounts

To configure AWS IAM Identity Center Accounts, please read the instructions detailed in Create Users with AWS IAM Identity Center.

Here's an example identity that is ready to be used for SSO login:



This account is also setup with full privileges so that it can access SQS resources:



Configure a SQS Queue

To configure an SQS Queue, please follow the instructions detailed in Create Amazon SQS Queue

Here's an example of a SQS Queue:

Download and Install Amazon SQS Connector Version 2 from Payara

To download and install Amazon SQS Connector V2, please check install-connector-server of the Amazon SQS Versioning.

Install Amazon CLI Locally

To install the Amazon CLI toolset on you local environment, please check Amazon CLI getting started

You can verify that the CLI has been correctly installed with the following command:

```
>aws --version
aws-cli/2.8.11 Python/3.9.11 Windows/10 exe/AMD64 prompt/off
```

Setup Local Configuration

Next, you'll have to setup the Amazon IAM configuration file on your local machine to enable SSO authentication. Read detailed instructions on how to setup this configuration file on AWS SSO Credentials.

Here's a sample configuration file:

```
[default]
sso_start_url = https://d-9067bf6e6d.awsapps.com/start
sso_region = us-east-1
sso_account_id = <ACCOUNT_ID>
sso_role_name = <ROLE_NAME>
```

IMPORTANT

The `sso_start_url`, `sso_region` and `sso_role_name` are parameters associated with the SSO credentials for the user account identity. To get the `sso_account_id` find the instructions in Account ID.

Before proceeding, you'll have to set the URL of the SQS Queue and its corresponding region as Environment Variables:

```
>set queueURL=https://sqs.us-east-1.amazonaws.com/845692786875/MyTestQueue
>set region=us-east-1
```

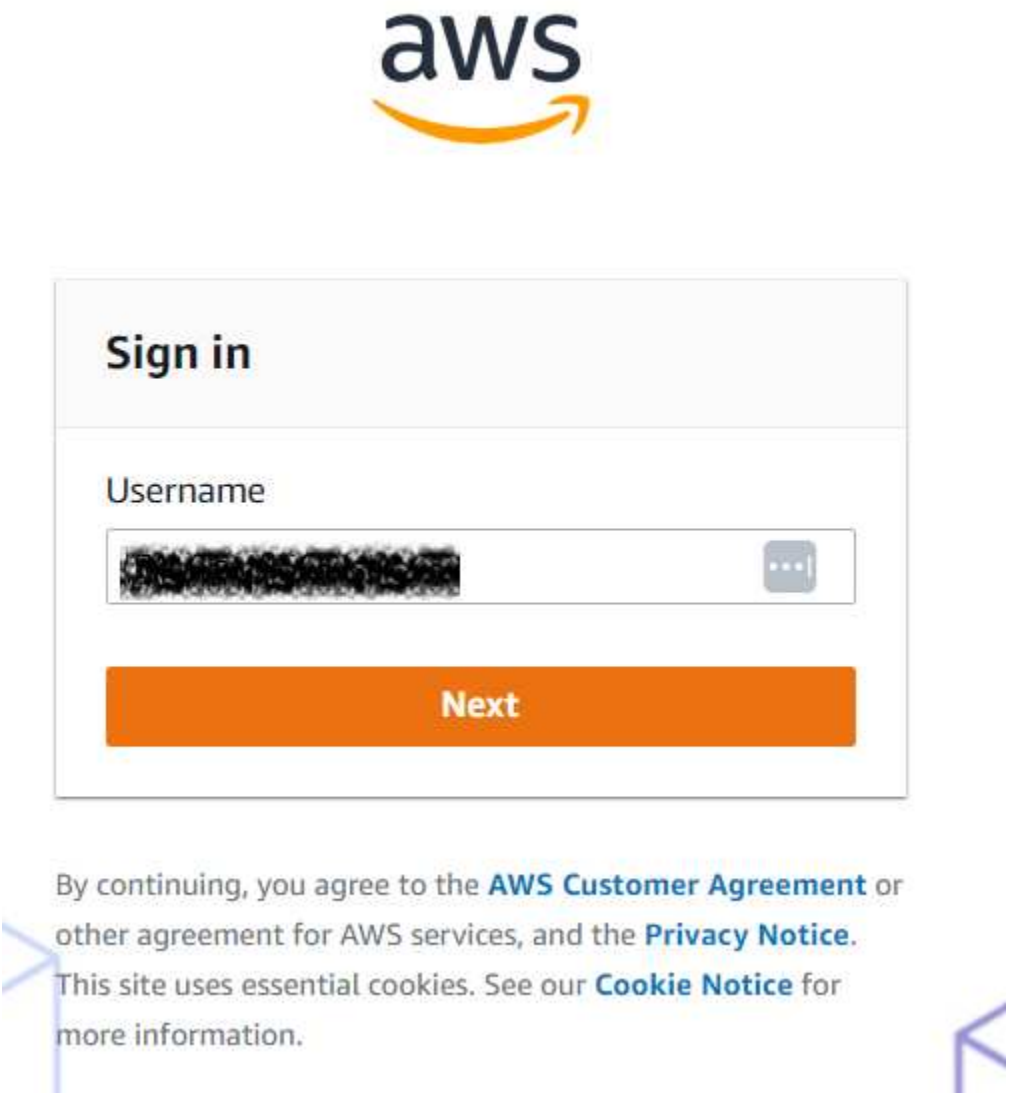
AWS SSO Login

Next, execute the AWS SSO login from the command line:

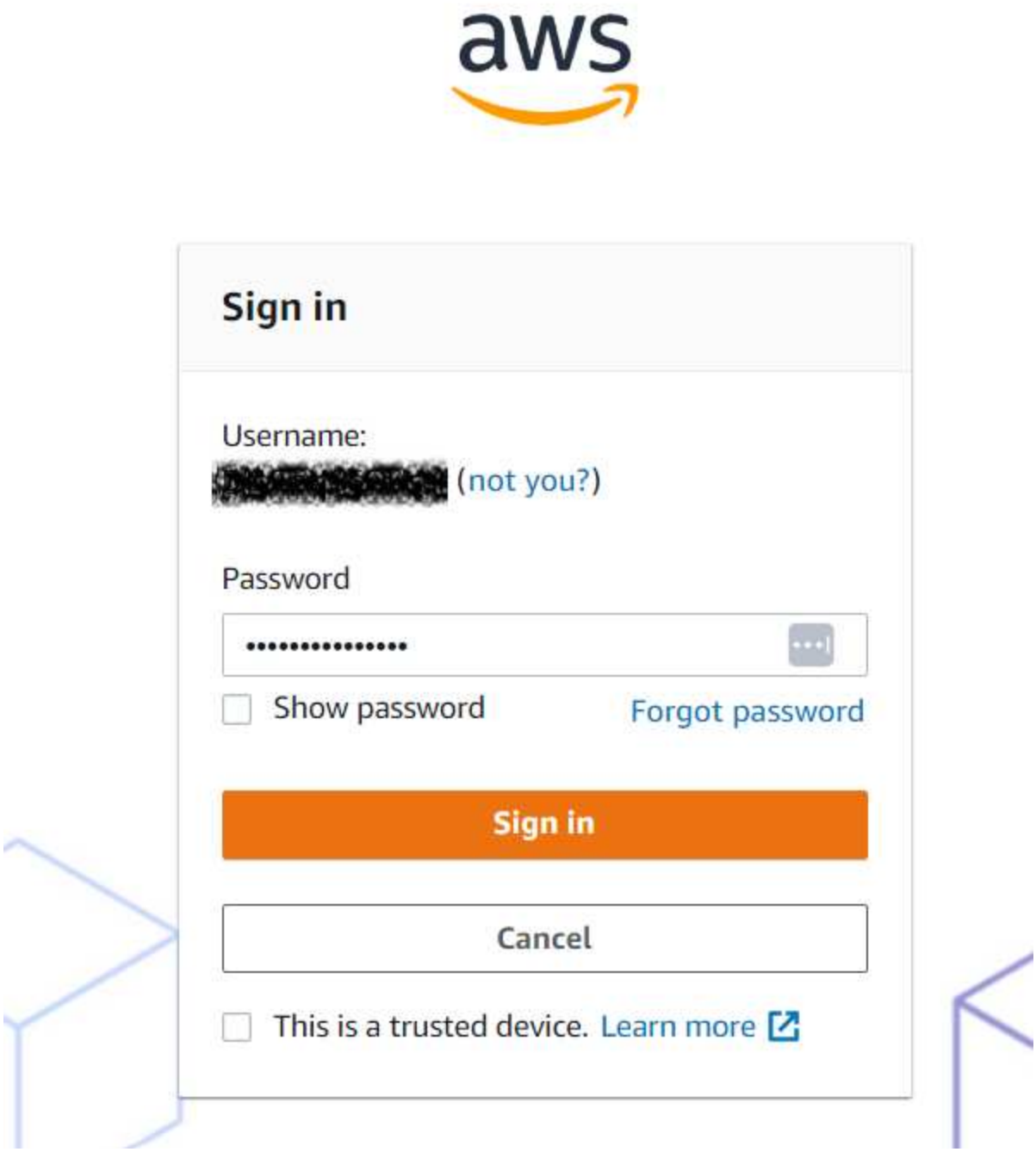
```
>aws sso login --profile default

Attempting to automatically open the SSO authorization page in your default
browser.
If the browser does not open or you wish to use a different device to authorize
this request, open the following URL:

https://device.sso.us-east-1.amazonaws.com/

Then enter the code:

XXXX-XXXX

Successfully logged into Start URL: https://d-9067bf6e6d.awsapps.com/start
```

This process will open a local browser window so that the current user can provide its credentials using this simple process:

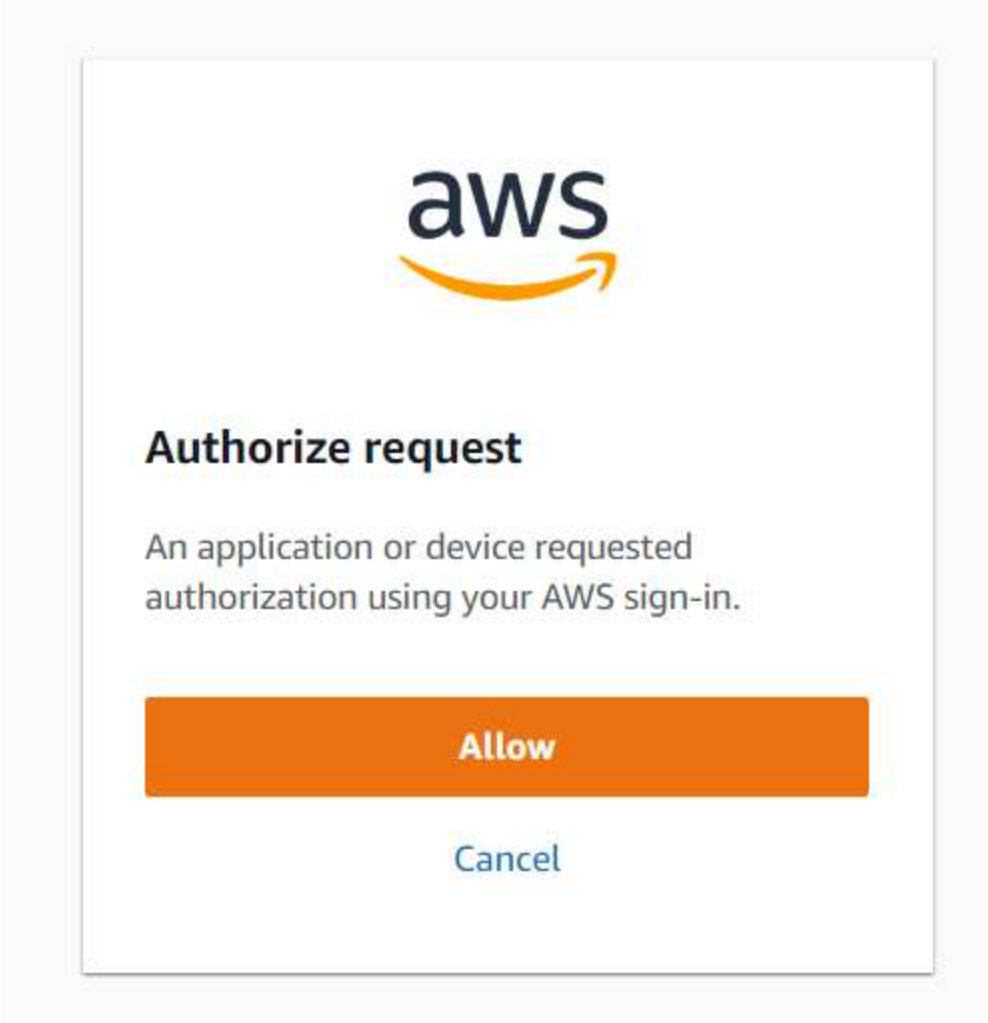
1. Start the session



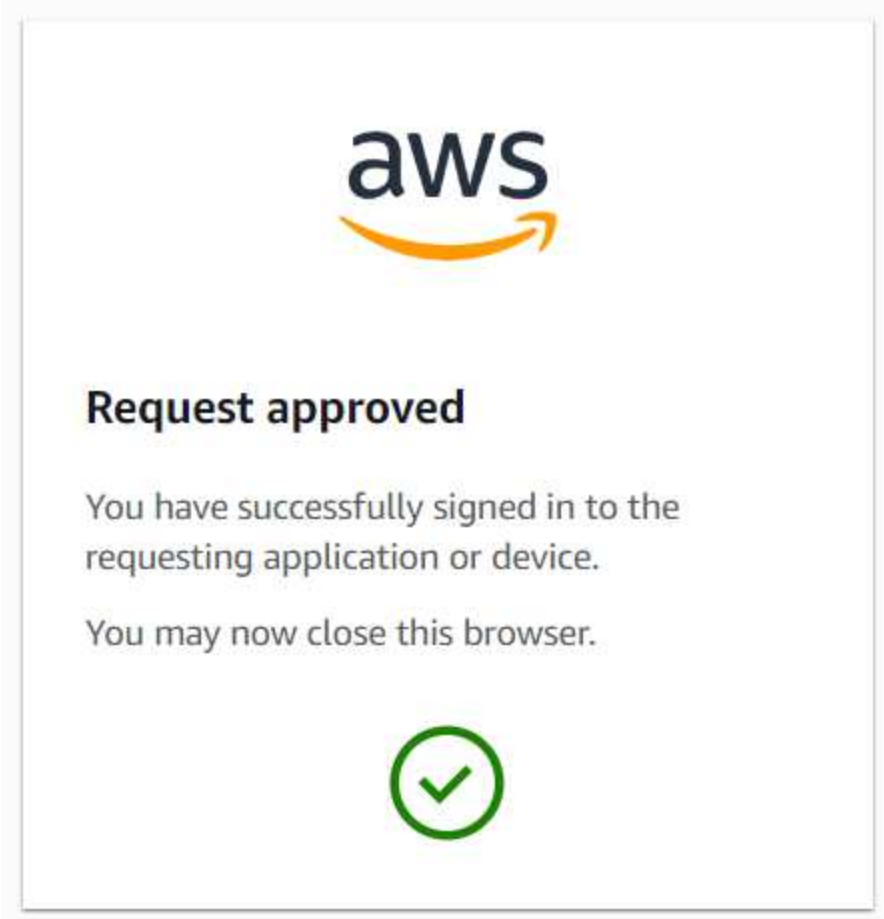
2. Set user password



3. Authorize the request



4. Receive login approval



Deploy and Test Application

Before deploying and testing a sample application, remember to install the JCA adapter and configuring the API dependencies as described in "Versioning".

Since the internal SSO credentials resolver will look for the required credentials on the local environment's configuration file, then you don't have to supply the AWS identity credentials on the configuration properties of your application's messaging components.

To send messages, here's how the connection factory definition looks like:

```
@ConnectionFactoryDefinition (
  name = "java:app/amazonsqs/factory",
  interfaceName =
"fish.payara.cloud.connectors.amazonsqs.api.AmazonSQSConnectionFactory",
  resourceAdapter = "amazon-sqs-rar-2.1.0",
  properties = {"region=us-east-1"}
)
```

And for receiving messages, here's how the MDB definition looks like:

```
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "queueURL", propertyValue =
"ENV={queueURL}"),
    @ActivationConfigProperty(propertyName = "region", propertyValue =
"ENV={region}"),
    @ActivationConfigProperty(propertyName = "pollInterval", propertyValue = "1")
})
public class ReceiveSQSMessage implements AmazonSQSListener {

    @OnSQSMessage
    public void receiveMessage(Message message) {
        // Handle message
    }
}
```

A simple log output of message processing would look like this:

```
[2022-12-11T19:30:51.455-0600] [] [INFORMACIÈN] [] [PayaraMicro] [tid: _ThreadID=1 _ThreadName=main] [timeMillis: 1670808651455] [levelValue: 800] Payara Mi
cro  ${major_version} #badassmicrofish (build ${build.number}) ready in 8,621 (ms)

Got message Hello World
Got attribute ApproximateReceiveCount,1
Got attribute SentTimestamp,1670808649872
Got attribute SenderId,AROA4JZZX3C53NVHAFOB2:DevTestSSOUser
Got attribute ApproximateFirstReceiveTimestamp,1670808649883
Got message Hello World
Got attribute ApproximateReceiveCount,1
Got attribute SentTimestamp,1670808650968
Got attribute SenderId,AROA4JZZX3C53NVHAFOB2:DevTestSSOUser
Got attribute ApproximateFirstReceiveTimestamp,1670808651053
Got message Hello World
Got attribute ApproximateReceiveCount,1
Got attribute SentTimestamp,1670808652072
Got attribute SenderId,AROA4JZZX3C53NVHAFOB2:DevTestSSOUser
Got attribute ApproximateFirstReceiveTimestamp,1670808652073
Got message Hello World
Got attribute ApproximateReceiveCount,1
Got attribute SentTimestamp,1670808653161
Got attribute SenderId,AROA4JZZX3C53NVHAFOB2:DevTestSSOUser
Got attribute ApproximateFirstReceiveTimestamp,1670808653170
Got message Hello World
```

Amazon Web Services STS Integration

To enhance security and provide temporary, limited-access credentials for your Jakarta EE application using Payara Cloud Connector for AWS SQS, you can integrate with AWS Security Token Service (STS) through Identity and Access Management (IAM). Follow the steps below to set up STS integration:

Creating IAM Role

To integrate Payara Cloud Connector for AWS SQS with your Jakarta EE application, you need to create an IAM (Identity and Access Management) role in the AWS Management Console. Follow the steps below to create an IAM role, retrieve the `roleArn`, and set a `roleSessionName` for use with Payara Cloud Connector AWS SQS.

Steps

1. Open the IAM Console
    a. Navigate to the AWS Management Console and open the IAM console.
2. Retrieve User ARN
    a. Navigate back to the IAM console.
    b. In the left navigation pane, choose "**Users**."
    c. Select the IAM user (e.g., "MY-USER").
    d. Copy the User ARN from the user's summary page (e.g., "arn:aws:iam::xxxxxxxxx:user/MY-USER").
3. Set Role Session Name
    a. For the `roleSessionName`, choose a logical name for your session (e.g., "PayaraSQSSession").
4. Create an IAM Role
    a. In the left navigation pane, choose "**Roles**".
    b. Choose "**Create role**".
    c. Select "**AWS account**".
    d. Enter the AWS account ID, If you're working within the same AWS account, you can skip this step.
    e. Choose "**Next**".
5. Add permissions
    a. Search for **AmazonSQSFullAccess** policiy and attach it.
    b. Choose "**Next**".
6. Role details
    a. Give the role a Name (e.g., "PayaraSQSRole").
    b. Add a meaningful description.

---

c. Choose "**Create role**".

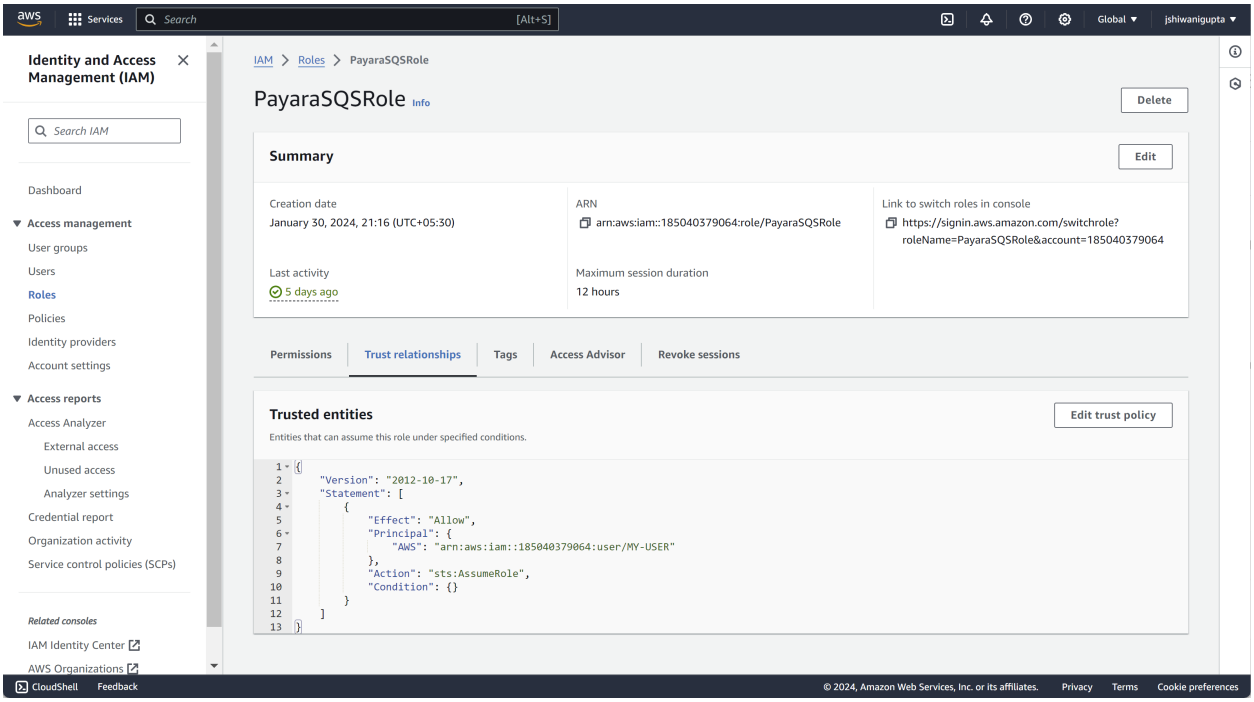
7. Retrieve Role ARN

   a. After creating the role, you'll see a summary page for the role. Copy the Role ARN (e.g., "arn:aws:iam::xxxxxxxxx:role/PayaraSQSRole").

8. Update Trust Relationship

To resolve potential authorization issues related to role assumption, ensure that the Trust Relationship in the IAM role is configured correctly. Update the Trust Relationship to explicitly allow the user to assume the role. Below is an example Trust Relationship JSON where root is replace with USER's ARN:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::xxxxxxxxx:user/MY-USER"
            },
            "Action": "sts:AssumeRole",
            "Condition": {}
        }
    ]
}
```



Make sure to replace the values with your specific IAM user ARN and role ARN.

It's crucial to establish a link between the IAM user's ARN and the IAM role's Trust Relationship by configuring the Trust Relationship with the user's ARN. This linkage ensures that the IAM user has the necessary permissions to assume the role. Additionally, for seamless integration, configure the `roleArn` and `roleSessionName` in the Payara Cloud Connector AWS SQS:

- `roleArn`: The ARN of the IAM role you've just created.
- `roleSessionName`: A logical name for your session (e.g., "PayaraSQSSession").

These values play a crucial role in the configuration of Payara Cloud Connector AWS SQS, enabling your Jakarta EE application to establish the necessary connections securely.

Sending Messages

Sending messages to Amazon SQS can be done via the JCA and an Amazon specific API. In order to start using this API to send messages, a resource has to be defined via the JCA API; a connection factory.

The connection factory has to be given a name, which can be any name that is valid for JNDI. The `java:app` namespace is typically recommended to be used. The type of the connection factory to be used for Amazon SQS is `fish.payara.cloud.connectors.amazonsqs.api.AmazonSQSConnectionFactory`, and we have to specify the resource adapter name which is here `amazon-sqs-rar-2.1.0.rar`.

Finally, the AWS account details have to be specified in the properties of the corresponding JMS resource definitions, including the `roleArn` and `roleSessionName`.

See the following example:

```
@ConnectionFactoryDefinition (
  name = "java:app/amazonsqs/factory",
```

```
  interfaceName =
"fish.payara.cloud.connectors.amazonsqs.api.AmazonSQSConnectionFactory",
  resourceAdapter = "amazon-sqs-rar-2.1.0",
  properties = {
        "awsAccessKeyId=<accessKeyID>",
        "awsSecretKey=<secretKey>",
        "roleArn=arn:aws:iam::xxxxxxxxx:role/PayaraSQSRole",
        "roleSessionName=PayaraSQSSession",
        "region=eu-west-2"
    }
)
```

With the above definition in place the following code shows an example of sending a message:

```
@Singleton
@Startup
public class SendSQSMessage {

 @Resource(lookup = "java:app/amazonsqs/factory")
 private AmazonSQSConnectionFactory factory;

 @PostConstruct
 public void init() {
    try (AmazonSQSConnection connection = factory.createConnection()) {
        SendMessageRequest sendMsgRequest = SendMessageRequest.builder()
                        .queueUrl("<queueURL>")
                        .messageBody("Hello World")
                        .build();
        connection.sendMessage(sendMsgRequest);
    }
    catch (Exception ex) {
    }
 }
}
```

Receiving Messages

Messages can be received from Amazon SQS by creating a MDB (Message Driven Bean) that implements the `fish.payara.cloud.connectors.amazonsqs.api.AmazonSQSListener` marker interface and has a single method annotated with `@OnSQSMessage` and the method signature `void method(Message message)`.

See the following example:

```
@MessageDriven(activationConfig = {
 @ActivationConfigProperty(propertyName = "awsAccessKeyId", propertyValue =
"someKey"),
```

```
 @ActivationConfigProperty(propertyName = "awsSecretKey", propertyValue =
"someSecretKey"),
 @ActivationConfigProperty(propertyName = "queueURL", propertyValue =
"someQueueURL"),
 @ActivationConfigProperty(propertyName = "pollInterval", propertyValue = "1"),
 @ActivationConfigProperty(propertyName = "roleArn", propertyValue =
"arn:aws:iam::xxxxxxxxx:role/PayaraSQSRole") ,
 @ActivationConfigProperty(propertyName = "roleSessionName", propertyValue =
"PayaraSQSSession") ,
 @ActivationConfigProperty(propertyName = "region", propertyValue = "eu-west-2")
})
public class ReceiveSQSMessage implements AmazonSQSListener {

 @OnSQSMessage
 public void receiveMessage(Message message) {
     // Handle message
 }
}
```

Programmatic SQS Queue Management

Amazon SQS Queues (and other resources) can be managed through the use of the official Amazon SQS API, which allows programmatic control over them. In general, managing queues in Amazon SQS is crucial for effective message handling.

To simplify the setup needed to use the SQS API from a standard Jakarta EE application and manage SQS resources, the Azul Payara SQS connector offers easy access to the `com.amazonaws.services.sqs.AmazonSQSConnectionFactory` class through the Java Connector Architecture (JCA) API.

This section will show how to access these facilities and focus on key operations like creating, listing, and deleting queues.

Connection Factory Setup

To utilize the Amazon SQS API for managing SQS resources in a Jakarta EE application, you must define a connection factory using the JCA API (via the `@ConnectionFactoryDefinition` annotation). This involves specifying the connection factory type and resource adapter name.

Additionally, AWS account details are required in the properties of the corresponding JMS resource definitions.

Once the connection factory is defined within the application, Jakarta EE components

can access it via resource injection to a `com.amazonaws.services.sqs.AmazonSQSConnectionFactory` typed object.

Here's a complete example of a singleton bean that defines the connection factory under the `java:app/amazonsqs/factory` JNDI name, which is used later to located it for resource injection under the bean's attributes:

```
@ConnectionFactoryDefinition (
    name = "java:app/amazonsqs/factory",
    interfaceName =
"fish.payara.cloud.connectors.amazonsqs.api.AmazonSQSConnectionFactory",
    resourceAdapter = "amazon-sqs-rar-2.1.0",
    properties = {"awsAccessKeyId=<accessKeyID>", "awsSecretKey=<secretKey>",
"region=<aws-region>"}
)
@Singleton
public class ManageSQSQueueBean {

    @Resource(lookup = "java:app/amazonsqs/factory")
    private AmazonSQSConnectionFactory factory;

    @PostConstruct
    public void listQueues() {
        try (AmazonSQSConnection connection = factory.createConnection()) {
            // Manage queues using AmazonSQSConnection
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

In the above example, The `@ConnectionFactoryDefinition` annotation sets up the connection factory, where properties like `awsAccessKeyId`, `awsSecretKey`, and `region` should be configured.

Within the `try` block, an instance of `AmazonSQSConnection` instance is created using the connection factory method (`factory.createConnection()`).

Management Operations

Once an `com.amazonaws.services.sqs.AmazonSQSConnection` instance is retrieved, it can be used to perform various queue management operations, such as creating a new queue, listing all queues, and deleting an existing queue.

Create Queue

Creating a new Amazon SQS queue is a fundamental operation. The following code snippet demonstrates how to create a queue:

```
@Singleton
public class ManageSQSQueueBean {

    public void createQueue(String name){
        try (var connection = factory.createConnection()) {
            var createQueueRequest = CreateQueueReque.builder()
                .queueName(name)
                .build();

            var createQueueResponse = connection.createQueue(createQueueRequest);
            System.out.println("Queue Created: " + createQueueResponse);
        }
    }
}
```

In this example, `MyNewQueue` is the name of the new queue. Additional parameters can be added to customize the queue creation.

List Queues

Listing queues provides insight into existing queues. The following code snippet demonstrates how to list all queues and those with a specific name prefix:

```
@Singleton
public class ManageSQSQueueBean {

    public void listAllQueues(){
        // List all queues
        try (var connection = factory.createConnection()) {
            var allQueues = connection.listQueues();
            System.out.println("All Queues:");
            for (var queueUrl : allQueues.queueUrls()) {
                System.out.println(" - " + queueUrl);
            }
        }
    }

    public void listQueues(String prefix){
        // List queues with a specific name prefix
        try (var connection = factory.createConnection()) {
            var prefixedQueues = connection.listQueues(ListQueuesRequest.builder
().queueNamePrefix(prefix).build());
            System.out.println("Prefixed Queues:");
            for (var queueUrl : prefixedQueues.queueUrls()) {
                System.out.println(" - " + queueUrl);
            }
        }
```

```
    }
}
```

Listing queues aids in monitoring and managing existing queues, allowing for targeted actions.

Delete Queue

Deleting an existing SQS queue is a critical operation. Ensure the complete queue URL is provided in the `DeleteQueueRequest`. The following code snippet illustrates how to delete a queue:

```
@Singleton
public class ManageSQSQueueBean {

    //Sample URL: https://sqs.<aws-region>.amazonaws.com/xxxxx/MyQueue
    public void delete(String url){
        try (var connection = factory.createConnection()) {
            var deleteQueueRequest = DeleteQueueRequest
                    .builder()
                    .queueUrl(url)
                    .build();

            var deleteQueueResponse = connection.deleteQueue(deleteQueueRequest);
            System.out.println("Queue Deleted: " + deleteQueueResponse);
        }
    }
}
```

CAUTION	Deleting a queue is an irreversible operation and results in the loss of all messages handled by the queue.

List Queue Tags

Listing queue tags provides metadata about a queue. The following code snippet demonstrates how to list tags for a specific queue:

```
@Singleton
public class ManageSQSQueueBean {

    public void listQueueTags(String url){
        try (var connection = factory.createConnection()) {
            ListQueueTagsRequest tagsRequest = ListQueueTagsRequest.builder()
                    .queueUrl(url)
                    .build();

            ListQueueTagsResponse tagsResponse = connection.listQueueTags
```

```
(tagsRequest);
            Map<String, String> tags = tagsResponse.tags();
            System.out.println("Queue Tags: " + tags);
        }
    }
}
```

Tags offer a way to categorize and organize queues based on specific attributes.

Purge Queue

Purging a queue deletes all messages within it. The following code snippet illustrates how to purge a queue:

```
@Singleton
public class ManageSQSQueueBean {

    public void purgeQueue(String url){
        try (var connection = factory.createConnection()) {
            var purgeRequest = PurgeQueueRequest.builder()
                    .queueUrl(url)
                    .build();

            var purgeResponse = connection.purgeQueue(purgeRequest);
            System.out.println("Queue Purged: " + purgeResponse);
        }
    }
}
```

CAUTION	Use this operation carefully, as it is irreversible, and it removes all messages from the specified queue.

Tag and Untag Queue

Adding and removing tags from a queue provides additional context. The following code snippet shows how to tag and untag a queue:

```
@Singleton
public class ManageSQSQueueBean {

    public void tagUntagQueue(String url){
        try (var connection = factory.createConnection()) {
            Map<String, String> tagsToAdd = Map.of("Environment", "Production",
"Team", "Dev");
            var tagQueueRequest = TagQueueRequest.builder()
                    .queueUrl(url)
                    .tags(tagsToAdd)
                    .build();
            var tagQueueResponse = connection.tagQueue(tagQueueRequest);
```

```
            System.out.println("Queue Tagged: " + tagQueueResponse);

            List<String> tagKeysToRemove = List.of("Environment");
            var untagQueueRequest = UntagQueueRequest.builder()
                    .queueUrl(url)
                    .tagKeys(tagKeysToRemove)
                    .build();
            var untagQueueResponse = connection.untagQueue(untagQueueRequest);
            System.out.println("Queue Untagged: " + untagQueueResponse);
        }
    }
}
```

TIP	Tagging allows for easy classification, while untagging removes unnecessary metadata.

Effectively managing Amazon SQS queues using the JCA API empowers developers to streamline message processing and maintain a well-organized messaging system.

See Also

For more detailed information on the Amazon SQS Java SDK, including additional features and advanced usage, refer to the official Javadocs.

Apache Kafka Cloud Connector

Apache Kafka is an open-source stream processing platform. It is based on a massively scalable publish/subscribe message queue architected as a distributed transaction log.

Versioning

The tags below are for the Cloud Connectors repository. You should use the one corresponding to the Jakarta version you require.

- 0.8.0: Kafka Connector using JakartaEE 8 and the `javax` namespace
- Kafka-1.0.0: Kafka Connector using JakartaEE 10 and the `jakarta` namespace

Usage

IMPORTANT	If you have selected to use Kafka Connector 1.0.0, replace all occurunces of `0.8.0` with `1.0.0` for the following example.

In order to connect to Apache Kafka, the `kafka-rar-0.8.0.rar` has to be deployed

as shown in the installing-a-connector of the Cloud Connectors overview.

Once deployed, something like the following should be printed to the startup log of Payara Micro:

```
[fish.payara.cloud.connectors.kafka.inbound.KafkaResourceAdapter] [tid: _ThreadID=1
_ThreadName=main] [timeMillis: 1495395212347] [levelValue: 800] Kafka Resource
Adapter Started..
```

In order to make use of this connector in an application, the following Maven dependency is needed:

```
<dependency>
  <groupId>fish.payara.cloud.connectors.kafka</groupId>
  <artifactId>kafka-jca-api</artifactId>
  <version>0.8.0</version>
  <type>jar</type>
  <scope>provided</scope>
</dependency>
```

Note that this dependency has scope `provided` since the types within this dependency are globally available to every application deployed to Payara Micro after the `kafka-rar-0.8.0.rar` was deployed.

Sending Messages

Sending messages to Apache Kafka can be done via the JCA and a Kafka specific API. In order to start using this API to send messages, a resource has to be defined via the JCA API; a connection factory.

The connection factory has to be given a name, which can be any name that is valid for JNDI. The java:app namespace is typically recommended to be used. The type of the connection factory to be used for Kafka is `fish.payara.cloud.connectors.kafka.KafkaConnectionFactory`, and we have to specify the resource adapter name which is here `kafka-rar-0.8.0`.

The following gives an example:

```
@ConnectionFactoryDefinition (
    name = "java:app/kafka/factory",
```

```
    interfaceName = "fish.payara.cloud.connectors.kafka.KafkaConnectionFactory",
    resourceAdapter = "kafka-rar-0.8.0"
)
```

With the above shown definition in place the following code shows an example of sending a message:

```
@Singleton
@Startup
public class SendKafkaMessage {

 @Resource(lookup = "java:app/kafka/factory")
 private KafkaConnectionFactory factory;

 @PostConstruct
 public void init() {
    try (KafkaConnection connection = factory.createConnection()) {
        connection.send(new ProducerRecord("test","hello","world"));
    }
    catch (Exception ex) {
    }
 }
}
```

Receiving Messages

Messages can be received from Apache Kafka by creating an MDB (Message Driven Bean) that implements the `fish.payara.cloud.connectors.kafka.api.KafkaListener` marker interface and has one or more methods annotated with `@OnRecord` and the method signature `void method(ConsumerRecord record)`, or `@OnRecords` and the method signature `void method(ConsumerRecords records)`

The following gives an example:

```
@MessageDriven(activationConfig = {
 @ActivationConfigProperty(propertyName = "groupIdConfig", propertyValue =
"testGroup"),
 @ActivationConfigProperty(propertyName = "topics", propertyValue = "test,test2"),
 @ActivationConfigProperty(propertyName = "autoCommitInterval", propertyValue =
"100"),
 @ActivationConfigProperty(propertyName = "pollInterval", propertyValue = "3000")
})
public class ReceiveKafkaMessage implements KafkaListener {

 @OnRecord(topics="test")
 public void receiveMessage(ConsumerRecord record) {
     // Handle record
```

```
  }
}
```

The full list of config properties is given below:

Config Property Name	Type	Default	Notes
`bootstrapServersConfig`	String	localhost:9092	Kafka Servers to connect to
`clientId`	String	KafkaJCAClient	Client ID of the Producer
`valueSerializer`	String	`org.apache.kafka.common.serialization.StringSerializer`	Serializer class for value
`keySerializer`	String	`org.apache.kafka.common.serialization.StringSerializer`	Serializer class for key
`bufferMemory`	Long	33554432	The total bytes the producer can use to buffer messages
`acks`	String	1	The number of acks the producer requires

Config Property Name	Type	Default	Notes
`retries`	Integer	0	The number of retries if there is a transient error
`batchSize`	Long	16384	The producer will attempt to batch records together into fewer requests whenever multiple records are being sent to the same partition
`lingerMS`	Long	0	The producer groups together any records that arrive in between request transmissions into a single batched request.
`maxBlockMS`	Long	60000	How long can send block
`maxRequestSize`	Long	1048576	Maximum size of request (bytes)
`receiveBufferBytes`	Integer	32768	Receive Buffer (bytes)
`requestTimeout`	Integer	30000	Request Timeout (ms)
`compression`	String	"none"	Compression type of data sent
`connectionsMaxIdle`	Long	540000	Close Idle Kafka Connections
`maxInflightConnections`	Integer	5	Maximum unacknowledged requests to send before blocking
`metadataMaxAge`	Long	300000	Period of time before a refresh of Metadata (ms)
`retryBackoff`	Long	100	The amount of time to wait before attempting a retry (ms)
`reconnectBackoff`	Long	100	The amount of time to wait before attempting a reconnection (ms)

Azure Service Bus Cloud Connector

The Microsoft Azure Service Bus allows code that runs both outside and inside Azure to communicate with Azure.

Versioning

The tags below are for the Cloud Connectors repository. You should use the one corresponding to the Jakarta version you require.

- 0.8.0: Azure Service Bus Connector using JakartaEE 8 and the `javax` namespace
- AzureServiceBus-1.0.0: Azure Service Bus Connector using JakartaEE 10 and the `jakarta` namespace

Usage

IMPORTANT	If you have selected to use Azure Service Bus Connector 1.0.0, replace all occurunces of `0.8.0` with `1.0.0` for the following example.

In order to connect to Azure, the `azure-sb-rar-0.8.0.rar` has to be deployed as shown in the installing-a-connector of the Cloud Connectors overview.

In order to make use of this connector in an application, the following maven dependency is needed:

```
<dependency>
  <groupId>fish.payara.cloud.connectors</groupId>
  <artifactId>azure-sb-jca-api</artifactId>
  <version>0.8.0</version>
  <type>jar</type>
  <scope>provided</scope>
</dependency>
```

Note that this dependency have scope provided since the types within this dependency are globally available to every application deployed to Payara Micro after the `azure-sb-rar-0.8.0.rar` was deployed.

Sending Messages

Sending messages to Azure can be done via the JCA and an Azure specific API. In order to start using this API to send messages, a resource has to be defined via the JCA API; a connection factory.

The connection factory has to be given a name, which can be any name that is valid for JNDI. The `java:app` namespace is typically recommended to be used. The type of the connection factory to be used for the Azure Service Bus is `fish.payara.cloud.connectors.azuresb.api.AzureSBConnectionFactory`

, and we have to specify the resource adapter name which is here `AzureSBRAR-0.1.0-SNAPSHOT`.

Finally the Azure credentials have to be specified.

The following gives an example:

```
@ConnectionFactoryDefinition (
  name = "java:app/azuresb/factory",
  interfaceName =
"fish.payara.cloud.connectors.azuresb.api.AzureSBConnectionFactory",
  resourceAdapter = "azure-sb-rar-0.8.0"
  properties = {"nameSpace=payara", "sasKeyName=RootManageSharedAccessKey",
"sasKey=someKey"}
)
```

With the above shown definition in place the following code shows an example of sending a message:

```
@Singleton
@Startup
public class SendAzureMessage {

  @Resource(lookup = "java:app/azuresb/factory")
  private AzureSBConnectionFactory factory;

  @PostConstruct
  public void init() {
    try (AzureSBConnection connection = factory.createConnection()) {
        connection.sendMessage("testq", new BrokeredMessage("Hello World"));
    }
    catch (Exception ex) {
    }
  }
}
```

Receiving Messages

Messages can be received from Azure by creating an MDB (Message Driven Bean) that implements the `fish.payara.cloud.connectors.azuresb.api.AzureSBListener` marker interface and has a single method annotated with `@OnAzureSBMessage` and the method signature `void method(BrokeredMessage message)`.

The following gives an example:

```
@MessageDriven(activationConfig = {
 @ActivationConfigProperty(propertyName = "nameSpace", propertyValue = "payara"),
 @ActivationConfigProperty(propertyName = "sasKeyName", propertyValue =
"RootManageSharedAccessKey"),
 @ActivationConfigProperty(propertyName = "sasKey", propertyValue = "someKey"),
 @ActivationConfigProperty(propertyName = "queueName", propertyValue = "testq")
})
public class ReceiveAzureMessage implements AzureSBListener {
    @OnAzureSBMessage
    public void receiveMessage(BrokeredMessage message) {
        // Handle message
    }
}
```

The full list of config properties is given below:

Config Property Name	Type	Default	Notes
sasKeyName	String	none	The SAS Key Name defined in your Service Bus namespace
sasKey	String	none	The SAS Key. Environment variable replacement can be used so that this is not exposed in the code
nameSpace	String	none	The Azure namespace of your Service Bus
queueName	String	none	The Queue Name (MDB property only)
initialPollDelay	Integer	1	The Initial Poll Delay (in seconds) before the Adapter starts polling for messages after deployment (MDB Property Only)
pollInterval	Integer	1	The Poll interval (in seconds). This is how often the MDB polls for new messages (MDB Property Only)
pollTimeout	Integer	1	The Poll Timeout (in seconds). This is how long the MDB should wait for messages in a single poll (MDB Property Only)

MQTT Cloud Connector

MQTT (Message Queue Telemetry Transport) is a lightweight publish-subscribe messaging protocol.

Versioning

The tags below are for the Cloud Connectors repository. You should use the one corresponding to the Jakarta version you require.

- 0.8.0: MQTT Connector using JakartaEE 8 and the `javax` namespace
- MQTT-1.0.0: Azure Service Bus Connector using JakartaEE 10 and the `jakarta` namespace

Usage

IMPORTANT	If you have selected to use MQTT Connector 1.0.0, replace all occurunces of `0.8.0` with `1.0.0` for the following example.

In order to connect to a MQTT broker, the `mqtt-rar-0.8.0.rar` has to be deployed as shown in the installing-a-connector of the Cloud Connectors overview.

In order to make use of this connector in an application, the following Maven dependency is needed:

```
<dependency>
  <groupId>fish.payara.cloud.connectors.MQTT</groupId>
  <artifactId>mqtt-jca-api</artifactId>
  <version>0.8.0</version>
  <type>jar</type>
  <scope>provided</scope>
</dependency>
```

Note that this dependency have scope provided since the types within this dependency are globally available to every application deployed to Payara Micro after the `mqtt-rar-0.8.0.rar` was deployed.

Sending Messages

Sending messages to a MQTT broker can be done via the JCA and an MQTT specific API. In order to start using this API to send messages, a resource has to be defined via the JCA API; a connection factory.

The connection factory has to be given a name, which can be any name that is valid for JNDI. The `java:app` namespace is typically recommended to be used. The type of the connection factory to be used for MQTT is `fish.payara.cloud.connectors.MQTT.api.MQTTConnectionFactory`, and we have to specify the resource adapter name which is here `mqtt-rar-0.8.0.rar`.

The following gives an example:

```
@ConnectionFactoryDefinition (
 name = "java:app/MQTT/factory",
 interfaceName = "fish.payara.cloud.connectors.MQTT.api.MQTTConnectionFactory",
 resourceAdapter = "mqtt-rar-0.8.0"
 properties = "cleanSession=true"
)
```

With the above shown definition in place the following code shows an example of sending a message:

```
@Singleton
@Startup
public class SendMQTTMessage {

 @Resource(lookup = "java:app/MQTT/factory")
 private MQTTConnectionFactory factory;

 @PostConstruct
 public void init() {
    try (MQTTConnection connection = factory.createConnection()) {
        connection.publish("test", "{\"test\": \"Hello World\"}".getBytes(), 0,
false);
    }
    catch (Exception ex) {
    }
  }
}
```

Receiving Messages

Messages can be received from an MQTT broker by creating an MDB (Message Driven Bean) that implements the `fish.payara.cloud.connectors.MQTT.api.MQTTListener` marker interface and has a single method annotated with `@OnMQTTMessage` and the method signature `void method(String topic, MqttMessage message)`.

The following gives an example:

```
@MessageDriven(activationConfig = {
 @ActivationConfigProperty(propertyName = "topicFilter", propertyValue = "test")
})
public class ReceiveMQTTMessage implements MQTTListener {
   @OnMQTTMessage
   public void receiveMessage(String topic, MqttMessage message) {
       // Handle message
```

```
    }
}
```

Config Property Name	Type	Default	Notes
`serverURIs`	String	tcp://localhost:1883	Server URIs for connection, comma separated
`cleanSession`	Boolean	false	Sets whether the client and server should remember state across reconnects
`automaticReconnect`	Boolean	true	Sets whether the client will automatically reconnect to the server if the connection is lost
`filePersistance`	Boolean	false	Whether the client should use file persistence for un-acknowledged messages
`persistenceDirectory`	String	.	Directory to use for file persistence
`connectionTimeout`	Integer	30	Sets the connection timeout value in seconds
`maxInflight`	Integer	10	Sets the maximum messages that can be sent without acknowledgements
`keepAliveInterval`	Integer	60	Sets the keep alive interval in seconds
`userName`	String	None	The username for the connection.
`password`	String	None	The password for the connection.
`topicFilter`	String	None	Topic Filter (For MDBs only)
`qos`	String	0	Quality of Service for the subscription (For MDBs only)

Security Connectors

Security Connectors are a collection of Jakarta EE Security adapters that implements industry-standard security protocols, specifications and integrates with third-party security artifacts & components.

The available connectors on Maven central are:

"OpenId Connect Client"

Implements OpenId Connect standard protocol by implementing Jakarta EE Security HTTP Authentication Mechanism and Identity Store for authentication & authorization using `@OpenIdAuthenticationDefinition` annotation and also provides access to the user's profile info, Identity token, Access token, & Refresh token using CDI based injection.

Maven Coordinates: `fish.payara.security.connectors:security-connector-oidc-client`

"OAuth 2.0 Client"

Implements OAuth 2.0 standard protocol by implementing Jakarta EE Security HTTP Authentication Mechanism and Identity Store for authorization using `@OAuth2AuthenticationDefinition` annotation and provides access to the Identity token and Access token using CDI based injection.

Maven Coordinates: `fish.payara.security.connectors:security-connector-oauth2-client`

Upgrade Advisor Tool

The Upgrade Advisor is a special Maven Plugin utility tool that offers developer advice about specific changes introduced in Jakarta EE 10 that affect Jakarta EE 8 applications and are necessary for these applications to be upgraded successfully.

The purpose of the tool is to summarize all changes that affect your application based on breaking changes introduced in the different Jakarta EE specifications by report log entries that indicate how the code of the application is affected and how it should be modified to be in full compliance of the Jakarta EE 10 specifications.

Usage

As with any other Maven plugins, the upgrade advisor can be run directly from the command line without any extra configuration required.

TIP	The tool is freely available in Maven Central.

To run the upgrade advisor, you must go to the root of your project folder and execute the following command:

```
mvn fish.payara.advisor:advisor-maven-plugin:1.0:advise
```

The advisor then will output its summary details:

```
[INFO] Showing Advisories
[INFO] ***************
[INFO] Line of code: 14 | Expression: public String registerConfigProvider(String
s, Map map, String s1, String s2, String s3)
Source file: TestAuthConfigFactory.java
Jakarta Authentication 3.0 Issue # 87
 jakarta.security.auth.message.config.AuthConfigFactory.registerConfigProvider was
changed from
 String registerConfigProvider(String className, Map properties, String layer,
String appContext, String description)
 to
 String registerConfigProvider(String className, Map<String, String> properties,
String layer, String appContext, String description)
This issue won't affect your application.
 The recommendation is to take care of the generic version.
[INFO] Line of code: 20 | Expression: public Class[] getSupportedMessageTypes()
Source file: TestAuthModule.java
Jakarta Authentication 3.0 Issue # 87
 jakarta.security.auth.message.module.ServerAuthModule.getSupportedMessageTypes was
changed from
 Class[] getSupportedMessageTypes()
 to
 Class<?>[] getSupportedMessageTypes()
This issue won't affect your application.
 The recommendation is to take care of the generic version.
```

Structure of the Advisor's Summary

The issues reported by the advisor's summary output will follow this structure, in order:

1. On the first line:
    a. A standard message with the `[Message Type] [Line of code] [Expression]` format.

    Message Type: This can be either `INFO`, `WARNING` or `ERROR` **Line of Code**: Indicates the location of this occurrence in the code. **Expression**: The Java code expression or statement that triggered the occurrence of the advisory.

    For example:

- A method declaration
- A method call,
- An annotation property

  etc.

2. On the second line:
   - The source file where the occurrence of the issue is reported.
3. On the third line:
   - A brief summary of the issue reported.
4. On the fourth line:
   - A set of recommendations given by the advisor to resolve the issue.

The following are examples of the different severity levels of findings reported by the advisor tool:

Example INFO Message

```
[INFO] Line of code: 27 | Expression: public Map getMap()
Source file: TestMessageInfo.java
Jakarta Authentication 3.0 Issue # 87
 jakarta.security.auth.message.MessageInfo.getMap was changed from
 Map getMap();
 to
 Map<String, Object> getMap();
This issue won't affect your application.
 The recommendation is to take care of the generic version.
```

Example WARNING Message

```
[WARNING] Line of code: 7 | Expression: @JsonbProperty(nillable = true)
Source file: JsonBeanTest.java
Jakarta JSON Binding 3.0
 property nillable from the JsonbProperty annotation was deprecated.
This issue won't affect your application.
 and the recommendation is to stop use of the deprecated property nillable and
 start to use the JsonbNillable annotation.
```

Example ERROR Message

```
[ERROR] Line of code: 23 | Expression: myMethodExpression.isParametersProvided()
Source file: TestBatchlet.java
```

```
Jakarta Expression Language 5.0 issue 117
 MethodExpression.isParametersProvided() has been removed.
Your application won't work, please remove MethodExpression.isParametersProvided()
call.
```

Basic Showcase Example

To showcase a specific example of the analysis done by the upgrade advisor, consider the following code snippet:

```
package fish.payara.upgrade;

import jakarta.json.bind.annotation.JsonbProperty;

public class JsonBeanTest {

    @JsonbProperty(nillable = true)
    private String name;
}
```

In Jakarta EE 10 the Jakarta JSON Binding 3.0 API's `@JsonbProperty` annotation was modified: Its `nillable` attribute has been marked as deprecated.

The upgrade advisor will report it as such when analysing this code snippet and will present advice in what to do regarding this change:

```
[WARNING] Line of code: 7 | Expression: @JsonbProperty(nillable = true)
Source file: JsonBeanTest.java
Jakarta JSON Binding 3.0
 property nillable from the JsonbProperty annotation was deprecated.
This issue won't affect your application.
 and the recommendation is to stop use of the deprecated property nillable and
 start to use the JsonbNillable annotation.
```

From the message we can see that the advisor recommends to use the new annotation `JsonbNillable` instead, like this:

```
package fish.payara.upgrade;

import jakarta.json.bind.annotation.JsonbNillable;
import jakarta.json.bind.annotation.JsonbProperty;

public class JsonBeanTest {

    @JsonbProperty
    @JsonbNillable
    private String name;
```

```
}
```

Developers are encouraged to follow the recommendations given by the advisor to circumvent most of the challenges and pain points involved in the upgrade of any application to Jakarta EE 10.

Miscellaneous

JAX-RS Extension

The JAX-RS Extension is an application specific extension that can assist you in improving the coding of certain aspects of any application that uses the standard JAX-RS Client API. At the moment the following features are supported:

- Define the `alias` of SSL certificate entry that will be used to establish secure communication in a mutual TLS transport setting.

Installation

To install this extension, simply include its JAR file in the application's WAR file. Using Maven, it is as simple as declaring the dependency through its GAV coordinates:

```
<dependency>
  <groupId>fish.payara.ecosystem.jaxrs</groupId>
  <artifactId>rest-ssl-configuration</artifactId>
  <version>2.0</version>
</dependency>
```

Usage

SSL Certificate Alias Configuration

To configure the alias of the SSL certificate to be used in secure communications established by the client, the following system property must be defined when establishing a new REST client call through the use of the `jakarta.ws.rs.client.ClientBuilder` class:

Property	Description

`fish.payara.jaxrs.client.certificate.alias`
 	
`PayaraConstants.JAXRS_CLIENT_CERTIFICATE_ALIAS`
	The alias name of the certificate

NOTE The name of the property is defined as a constant in the `fish.payara.security.client.PayaraConstants` class.

IMPORTANT The certificate that corresponds to the given alias must exist in the domain's configured keystore.

Example

Here's an example of how to set up the property when executing a new JAX-RS client call:

```
ClientBuilder.newBuilder()
             .property(PayaraConstants.JAXRS_CLIENT_CERTIFICATE_ALIAS,
"someAliasName")
             .build()
             .target(new URI("https://localhost:8080/service"))
             .request()
             .get(Service.class)
```

NOTE For this example to work, you need to include the certificate with the given alias in the default Payara Server keystore.

Dependencies

Payara Platform Dependencies

This page contains a detailed list of the dependencies used by the Payara Platform, along with the Maven GAV coordinates of each dependency.

NOTE Use this page to search the specific version of a dependency included in a Payara Platform distribution product.

- "Jakarta EE Specification Dependencies Mapping" - The dependencies related to the Jakarta EE specifications implemented by the Payara Platform.
- "MicroProfile Specification Dependencies Mapping" - The dependencies related to the Eclipse MicroProfile specifications implemented by the Payara Platform.
- "Internal Dependencies" - The internal dependencies used by the Payara Platform.

Jakarta EE Specification Dependencies Mapping

This section details all dependencies linked to Jakarta EE specifications.

activation

angus-activation

Version: 2.0.3

```
<dependency>
    <groupId>org.eclipse.angus</groupId>
    <artifactId>angus-activation</artifactId>
    <version>2.0.3</version>
</dependency>
```

jakarta.activation-api

Version: 2.1.4

```
<dependency>
    <groupId>jakarta.activation</groupId>
    <artifactId>jakarta.activation-api</artifactId>
    <version>2.1.4</version>
</dependency>
```

annotation

annotation-framework

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>annotation-framework</artifactId>
    <version>7.0.0</version>
</dependency>
```

jackson-annotations

Version: 2.21

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.21</version>
</dependency>
```

jackson-jr-annotation-support

Version: 2.15.2

```
<dependency>
    <groupId>com.fasterxml.jackson.jr</groupId>
    <artifactId>jackson-jr-annotation-support</artifactId>
    <version>2.15.2</version>
</dependency>
```

jackson-module-jakarta-xmlbind-annotations

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.module</groupId>
    <artifactId>jackson-module-jakarta-xmlbind-annotations</artifactId>
    <version>2.21.2</version>
</dependency>
```

jakarta.annotation-api

Version: 3.0.0

```
<dependency>
    <groupId>jakarta.annotation</groupId>
    <artifactId>jakarta.annotation-api</artifactId>
    <version>3.0.0</version>
</dependency>
```

jcip-annotations

Version: 1.0-1

```
<dependency>
    <groupId>com.github.stephenc.jcip</groupId>
    <artifactId>jcip-annotations</artifactId>
    <version>1.0-1</version>
```

```
</dependency>
```

jcip-annotations-repackaged

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>jcip-annotations-repackaged</artifactId>
    <version>7.0.0</version>
</dependency>
```

authentication

jakarta.authentication-api

Version: 3.1.0

```
<dependency>
    <groupId>jakarta.authentication</groupId>
    <artifactId>jakarta.authentication-api</artifactId>
    <version>3.1.0</version>
</dependency>
```

authorization

jacc.provider.inmemory

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
    <artifactId>jacc.provider.inmemory</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta.authorization-api

Version: 3.0.0

```
<dependency>
    <groupId>jakarta.authorization</groupId>
    <artifactId>jakarta.authorization-api</artifactId>
    <version>3.0.0</version>
</dependency>
```

jaspic-servlet-utils

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>jaspic-servlet-utils</artifactId>
    <version>7.0.0</version>
</dependency>
```

batch

glassfish-batch-commands

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.batch</groupId>
    <artifactId>glassfish-batch-commands</artifactId>
    <version>7.0.0</version>
</dependency>
```

glassfish-batch-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.batch</groupId>
    <artifactId>glassfish-batch-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta.batch-api

Version: 2.1.1

```
<dependency>
    <groupId>jakarta.batch</groupId>
    <artifactId>jakarta.batch-api</artifactId>
    <version>2.1.1</version>
</dependency>
```

payara-jbatch

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.batch</groupId>
    <artifactId>payara-jbatch</artifactId>
    <version>7.0.0</version>
</dependency>
```

concurrency

concurrent-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.concurrent</groupId>
    <artifactId>concurrent-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

concurrent-impl

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.concurrent</groupId>
    <artifactId>concurrent-impl</artifactId>
    <version>7.0.0</version>
</dependency>
```

console-concurrent-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-concurrent-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta.enterprise.concurrent-api

Version: 3.1.1

```
<dependency>
    <groupId>jakarta.enterprise.concurrent</groupId>
    <artifactId>jakarta.enterprise.concurrent-api</artifactId>
    <version>3.1.1</version>
</dependency>
```

connectors

connectors-admin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.connectors</groupId>
```

```
    <artifactId>connectors-admin</artifactId>
    <version>7.0.0</version>
</dependency>
```

connectors-inbound-runtime

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.connectors</groupId>
    <artifactId>connectors-inbound-runtime</artifactId>
    <version>7.0.0</version>
</dependency>
```

connectors-internal-api

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.connectors</groupId>
    <artifactId>connectors-internal-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

connectors-runtime

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.connectors</groupId>
    <artifactId>connectors-runtime</artifactId>
    <version>7.0.0</version>
</dependency>
```

gf-connectors-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.connectors</groupId>
    <artifactId>gf-connectors-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

resources-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.resources</groupId>
    <artifactId>resources-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

resources-runtime

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.resources</groupId>
    <artifactId>resources-runtime</artifactId>
    <version>7.0.0</version>
</dependency>
```

security-connectors-api

Version: 4.0.0.alpha3

```
<dependency>
    <groupId>fish.payara.security.connectors</groupId>
    <artifactId>security-connectors-api</artifactId>
    <version>4.0.0.alpha3</version>
</dependency>
```

context-dependency-injection

cdi-api-fragment

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>cdi-api-fragment</artifactId>
    <version>7.0.0</version>
</dependency>
```

cdi-auth-roles

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>cdi-auth-roles</artifactId>
    <version>7.0.0</version>
</dependency>
```

cdieventbus-notifier-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>cdieventbus-notifier-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

gf-weld-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>gf-weld-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

hibernate-validator-cdi

Version: 9.0.1.final.payara-p1

```
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator-cdi</artifactId>
    <version>9.0.1.final.payara-p1</version>
</dependency>
```

jakarta.enterprise.cdi-api

Version: 4.1.0

```
<dependency>
    <groupId>jakarta.enterprise</groupId>
    <artifactId>jakarta.enterprise.cdi-api</artifactId>
    <version>4.1.0</version>
</dependency>
```

jakarta.enterprise.cdi-el-api

Version: 4.1.0

```
<dependency>
    <groupId>jakarta.enterprise</groupId>
    <artifactId>jakarta.enterprise.cdi-el-api</artifactId>
    <version>4.1.0</version>
```

```
</dependency>
```

jakarta.inject-api

Version: 2.0.1

```
<dependency>
    <groupId>jakarta.inject</groupId>
    <artifactId>jakarta.inject-api</artifactId>
    <version>2.0.1</version>
</dependency>
```

jersey-cdi1x

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext.cdi</groupId>
    <artifactId>jersey-cdi1x</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-cdi1x-servlet

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext.cdi</groupId>
    <artifactId>jersey-cdi1x-servlet</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-cdi1x-transaction

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext.cdi</groupId>
    <artifactId>jersey-cdi1x-transaction</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

notification-cdi-eventbus-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
```

```
    <artifactId>notification-cdi-eventbus-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

payara-micro-cdi

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>payara-micro-cdi</artifactId>
    <version>7.0.0</version>
</dependency>
```

soteria.spi.bean.decorator.weld

Version: 4.0.1.payara-p1

```
<dependency>
    <groupId>org.glassfish.soteria</groupId>
    <artifactId>soteria.spi.bean.decorator.weld</artifactId>
    <version>4.0.1.payara-p1</version>
</dependency>
```

tyrus-container-glassfish-cdi

Version: 2.2.2

```
<dependency>
    <groupId>org.glassfish.tyrus</groupId>
    <artifactId>tyrus-container-glassfish-cdi</artifactId>
    <version>2.2.2</version>
</dependency>
```

weld-api

Version: 6.0.final

```
<dependency>
    <groupId>org.jboss.weld</groupId>
    <artifactId>weld-api</artifactId>
    <version>6.0.final</version>
</dependency>
```

weld-core-impl

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld</groupId>
    <artifactId>weld-core-impl</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-ejb

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld.module</groupId>
    <artifactId>weld-ejb</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-integration

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>weld-integration</artifactId>
    <version>7.0.0</version>
</dependency>
```

weld-integration-fragment

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>weld-integration-fragment</artifactId>
    <version>7.0.0</version>
</dependency>
```

weld-jsf

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld.module</groupId>
    <artifactId>weld-jsf</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-jta

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld.module</groupId>
    <artifactId>weld-jta</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-lite-extension-translator

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld</groupId>
    <artifactId>weld-lite-extension-translator</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-osgi-bundle

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld</groupId>
    <artifactId>weld-osgi-bundle</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-spi

Version: 6.0.final

```
<dependency>
    <groupId>org.jboss.weld</groupId>
    <artifactId>weld-spi</artifactId>
    <version>6.0.final</version>
</dependency>
```

weld-web

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld.module</groupId>
    <artifactId>weld-web</artifactId>
    <version>6.0.4.final</version>
```

```
</dependency>
```

data

data-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.data</groupId>
    <artifactId>data-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

datadog-notifier-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>datadog-notifier-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

datadog-notifier-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>datadog-notifier-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

dataprovider

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>dataprovider</artifactId>
    <version>7.0.0</version>
</dependency>
```

jackson-databind

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.21.2</version>
</dependency>
```

jackson-dataformat-toml

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-toml</artifactId>
    <version>2.21.2</version>
</dependency>
```

jackson-dataformat-xml

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.21.2</version>
</dependency>
```

jackson-dataformat-yaml

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-yaml</artifactId>
    <version>2.21.2</version>
</dependency>
```

jakarta.data-api

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>jakarta.data-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

relaxng-datatype

Version: 4.0.7

```
<dependency>
    <groupId>com.sun.xml.bind.external</groupId>
    <artifactId>relaxng-datatype</artifactId>
    <version>4.0.7</version>
</dependency>
```

dependency-injection

gf-jms-injection

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.jms</groupId>
    <artifactId>gf-jms-injection</artifactId>
    <version>7.0.0</version>
</dependency>
```

gf-weld-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>gf-weld-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta.inject-api

Version: 2.0.1

```
<dependency>
    <groupId>jakarta.inject</groupId>
    <artifactId>jakarta.inject-api</artifactId>
    <version>2.0.1</version>
</dependency>
```

soteria.spi.bean.decorator.weld

Version: 4.0.1.payara-p1

```
<dependency>
    <groupId>org.glassfish.soteria</groupId>
    <artifactId>soteria.spi.bean.decorator.weld</artifactId>
    <version>4.0.1.payara-p1</version>
```

```
</dependency>
```

weld-api

Version: 6.0.final

```
<dependency>
    <groupId>org.jboss.weld</groupId>
    <artifactId>weld-api</artifactId>
    <version>6.0.final</version>
</dependency>
```

weld-core-impl

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld</groupId>
    <artifactId>weld-core-impl</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-ejb

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld.module</groupId>
    <artifactId>weld-ejb</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-integration

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>weld-integration</artifactId>
    <version>7.0.0</version>
</dependency>
```

weld-integration-fragment

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
```

```
    <artifactId>weld-integration-fragment</artifactId>
    <version>7.0.0</version>
</dependency>
```

weld-jsf

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld.module</groupId>
    <artifactId>weld-jsf</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-jta

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld.module</groupId>
    <artifactId>weld-jta</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-lite-extension-translator

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld</groupId>
    <artifactId>weld-lite-extension-translator</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-osgi-bundle

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld</groupId>
    <artifactId>weld-osgi-bundle</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

weld-spi

Version: 6.0.final

```
<dependency>
    <groupId>org.jboss.weld</groupId>
    <artifactId>weld-spi</artifactId>
    <version>6.0.final</version>
</dependency>
```

weld-web

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld.module</groupId>
    <artifactId>weld-web</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

deployment

deployment-admin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.deployment</groupId>
    <artifactId>deployment-admin</artifactId>
    <version>7.0.0</version>
</dependency>
```

deployment-autodeploy

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.deployment</groupId>
    <artifactId>deployment-autodeploy</artifactId>
    <version>7.0.0</version>
</dependency>
```

deployment-client

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.deployment</groupId>
    <artifactId>deployment-client</artifactId>
    <version>7.0.0</version>
</dependency>
```

deployment-common

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.deployment</groupId>
    <artifactId>deployment-common</artifactId>
    <version>7.0.0</version>
</dependency>
```

deployment-javaee-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.deployment</groupId>
    <artifactId>deployment-javaee-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

deployment-javaee-full

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.deployment</groupId>
    <artifactId>deployment-javaee-full</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta.enterprise.deploy-api

Version: 1.7.2

```
<dependency>
    <groupId>jakarta.enterprise.deploy</groupId>
    <artifactId>jakarta.enterprise.deploy-api</artifactId>
    <version>1.7.2</version>
</dependency>
```

ee-platform

jakarta.enterprise.lang-model

Version: 4.1.0

```
<dependency>
    <groupId>jakarta.enterprise</groupId>
    <artifactId>jakarta.enterprise.lang-model</artifactId>
    <version>4.1.0</version>
```

```
</dependency>
```

jakarta.resource-api

Version: 2.1.0

```
<dependency>
    <groupId>jakarta.resource</groupId>
    <artifactId>jakarta.resource-api</artifactId>
    <version>2.1.0</version>
</dependency>
```

jakartaee-kernel

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.core</groupId>
    <artifactId>jakartaee-kernel</artifactId>
    <version>7.0.0</version>
</dependency>
```

enterprise-beans

console-ejb-lite-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-ejb-lite-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

console-ejb-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-ejb-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

ejb-client

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.ejb</groupId>
    <artifactId>ejb-client</artifactId>
    <version>7.0.0</version>
</dependency>
```

ejb-container

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.ejb</groupId>
    <artifactId>ejb-container</artifactId>
    <version>7.0.0</version>
</dependency>
```

ejb-full-container

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.ejb</groupId>
    <artifactId>ejb-full-container</artifactId>
    <version>7.0.0</version>
</dependency>
```

ejb-http-admin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.ejb</groupId>
    <artifactId>ejb-http-admin</artifactId>
    <version>7.0.0</version>
</dependency>
```

ejb-internal-api

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.ejb</groupId>
    <artifactId>ejb-internal-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

ejb-opentracing

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.ejb</groupId>
    <artifactId>ejb-opentracing</artifactId>
    <version>7.0.0</version>
</dependency>
```

ejb.security

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
    <artifactId>ejb.security</artifactId>
    <version>7.0.0</version>
</dependency>
```

gf-ejb-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.ejb</groupId>
    <artifactId>gf-ejb-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

hazelcast-ejb-timer

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>hazelcast-ejb-timer</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta.ejb-api

Version: 4.0.1

```
<dependency>
    <groupId>jakarta.ejb</groupId>
    <artifactId>jakarta.ejb-api</artifactId>
    <version>4.0.1</version>
```

```
</dependency>
```

jersey-gf-ejb

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.containers.glassfish</groupId>
    <artifactId>jersey-gf-ejb</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

schema2beans

Version: 6.7

```
<dependency>
    <groupId>org.glassfish.external</groupId>
    <artifactId>schema2beans</artifactId>
    <version>6.7</version>
</dependency>
```

tyrus-container-glassfish-ejb

Version: 2.2.2

```
<dependency>
    <groupId>org.glassfish.tyrus</groupId>
    <artifactId>tyrus-container-glassfish-ejb</artifactId>
    <version>2.2.2</version>
</dependency>
```

weld-ejb

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld.module</groupId>
    <artifactId>weld-ejb</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

`expression-language`

expressly

Version: 6.0.0

```
<dependency>
    <groupId>org.glassfish.expressly</groupId>
    <artifactId>expressly</artifactId>
    <version>6.0.0</version>
</dependency>
```

jakarta.el-api

Version: 6.0.1

```
<dependency>
    <groupId>jakarta.el</groupId>
    <artifactId>jakarta.el-api</artifactId>
    <version>6.0.1</version>
</dependency>
```

jakarta.enterprise.cdi-el-api

Version: 4.1.0

```
<dependency>
    <groupId>jakarta.enterprise</groupId>
    <artifactId>jakarta.enterprise.cdi-el-api</artifactId>
    <version>4.1.0</version>
</dependency>
```

faces

faces-compat

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>faces-compat</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta.faces

Version: 4.1.7.payara-p1

```
<dependency>
    <groupId>org.glassfish</groupId>
    <artifactId>jakarta.faces</artifactId>
    <version>4.1.7.payara-p1</version>
</dependency>
```

jsf-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>jsf-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

jsft

Version: 4.2.0

```
<dependency>
    <groupId>org.glassfish.jsftemplating</groupId>
    <artifactId>jsft</artifactId>
    <version>4.2.0</version>
</dependency>
```

jsftemplating

Version: 4.2.0

```
<dependency>
    <groupId>org.glassfish.jsftemplating</groupId>
    <artifactId>jsftemplating</artifactId>
    <version>4.2.0</version>
</dependency>
```

jsftemplating-dt

Version: 4.2.0

```
<dependency>
    <groupId>org.glassfish.jsftemplating</groupId>
    <artifactId>jsftemplating-dt</artifactId>
    <version>4.2.0</version>
</dependency>
```

weld-jsf

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld.module</groupId>
    <artifactId>weld-jsf</artifactId>
    <version>6.0.4.final</version>
```

```
</dependency>
```

interceptors

jakarta.interceptor-api

Version: 2.2.0

```
<dependency>
    <groupId>jakarta.interceptor</groupId>
    <artifactId>jakarta.interceptor-api</artifactId>
    <version>2.2.0</version>
</dependency>
```

json-binding

jakarta.json.bind-api

Version: 3.0.1

```
<dependency>
    <groupId>jakarta.json.bind</groupId>
    <artifactId>jakarta.json.bind-api</artifactId>
    <version>3.0.1</version>
</dependency>
```

jersey-media-json-binding

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-binding</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

yasson

Version: 3.0.4

```
<dependency>
    <groupId>org.eclipse</groupId>
    <artifactId>yasson</artifactId>
    <version>3.0.4</version>
</dependency>
```

json-processing

everit-json-schema

Version: 1.14.3

```
<dependency>
    <groupId>com.github.erosb</groupId>
    <artifactId>everit-json-schema</artifactId>
    <version>1.14.3</version>
</dependency>
```

jackson-annotations

Version: 2.21

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.21</version>
</dependency>
```

jackson-core

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.21.2</version>
</dependency>
```

jackson-databind

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.21.2</version>
</dependency>
```

jackson-dataformat-toml

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-toml</artifactId>
    <version>2.21.2</version>
```

```
</dependency>
```

jackson-dataformat-xml

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.21.2</version>
</dependency>
```

jackson-dataformat-yaml

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-yaml</artifactId>
    <version>2.21.2</version>
</dependency>
```

jackson-jr-annotation-support

Version: 2.15.2

```
<dependency>
    <groupId>com.fasterxml.jackson.jr</groupId>
    <artifactId>jackson-jr-annotation-support</artifactId>
    <version>2.15.2</version>
</dependency>
```

jackson-jr-objects

Version: 2.15.2

```
<dependency>
    <groupId>com.fasterxml.jackson.jr</groupId>
    <artifactId>jackson-jr-objects</artifactId>
    <version>2.15.2</version>
</dependency>
```

jackson-module-jakarta-xmlbind-annotations

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.module</groupId>
```

```
    <artifactId>jackson-module-jakarta-xmlbind-annotations</artifactId>
    <version>2.21.2</version>
</dependency>
```

jakarta.json

Version: 1.1.7.payara-p1

```
<dependency>
    <groupId>org.eclipse.parsson</groupId>
    <artifactId>jakarta.json</artifactId>
    <version>1.1.7.payara-p1</version>
</dependency>
```

jakarta.json-api

Version: 2.1.0

```
<dependency>
    <groupId>jakarta.json</groupId>
    <artifactId>jakarta.json-api</artifactId>
    <version>2.1.0</version>
</dependency>
```

jakarta.json.bind-api

Version: 3.0.1

```
<dependency>
    <groupId>jakarta.json.bind</groupId>
    <artifactId>jakarta.json.bind-api</artifactId>
    <version>3.0.1</version>
</dependency>
```

jersey-media-json-binding

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-binding</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-media-json-jackson

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-jackson</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-media-json-processing

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-processing</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

json

Version: 20231013

```
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20231013</version>
</dependency>
```

json-smart

Version: 2.6.0

```
<dependency>
    <groupId>net.minidev</groupId>
    <artifactId>json-smart</artifactId>
    <version>2.6.0</version>
</dependency>
```

parsson

Version: 1.1.7.payara-p1

```
<dependency>
    <groupId>org.eclipse.parsson</groupId>
    <artifactId>parsson</artifactId>
    <version>1.1.7.payara-p1</version>
</dependency>
```

parsson-media

Version: 1.1.7.payara-p1

```
<dependency>
    <groupId>org.eclipse.parsson</groupId>
    <artifactId>parsson-media</artifactId>
    <version>1.1.7.payara-p1</version>
</dependency>
```

mail

angus-activation

Version: 2.0.3

```
<dependency>
    <groupId>org.eclipse.angus</groupId>
    <artifactId>angus-activation</artifactId>
    <version>2.0.3</version>
</dependency>
```

angus-core

Version: 2.0.5

```
<dependency>
    <groupId>org.eclipse.angus</groupId>
    <artifactId>angus-core</artifactId>
    <version>2.0.5</version>
</dependency>
```

angus-mail

Version: 2.0.5

```
<dependency>
    <groupId>org.eclipse.angus</groupId>
    <artifactId>angus-mail</artifactId>
    <version>2.0.5</version>
</dependency>
```

email-notifier-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>email-notifier-console-plugin</artifactId>
    <version>7.0.0</version>
```

```
</dependency>
```

email-notifier-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>email-notifier-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

imap

Version: 2.0.5

```
<dependency>
    <groupId>org.eclipse.angus</groupId>
    <artifactId>imap</artifactId>
    <version>2.0.5</version>
</dependency>
```

jakarta.mail-api

Version: 2.1.5

```
<dependency>
    <groupId>jakarta.mail</groupId>
    <artifactId>jakarta.mail-api</artifactId>
    <version>2.1.5</version>
</dependency>
```

javamail-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.resources</groupId>
    <artifactId>javamail-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

javamail-runtime

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.resources</groupId>
```

```
    <artifactId>javamail-runtime</artifactId>
    <version>7.0.0</version>
</dependency>
```

logging-mailhandler

Version: 2.0.5

```
<dependency>
    <groupId>org.eclipse.angus</groupId>
    <artifactId>logging-mailhandler</artifactId>
    <version>2.0.5</version>
</dependency>
```

pop3

Version: 2.0.5

```
<dependency>
    <groupId>org.eclipse.angus</groupId>
    <artifactId>pop3</artifactId>
    <version>2.0.5</version>
</dependency>
```

smtp

Version: 2.0.5

```
<dependency>
    <groupId>org.eclipse.angus</groupId>
    <artifactId>smtp</artifactId>
    <version>2.0.5</version>
</dependency>
```

management

certificate-management-admin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.certificate-management</groupId>
    <artifactId>certificate-management-admin</artifactId>
    <version>7.0.0</version>
</dependency>
```

certificate-management-common

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.certificate-management</groupId>
    <artifactId>certificate-management-common</artifactId>
    <version>7.0.0</version>
</dependency>
```

certificate-management-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.certificate-management</groupId>
    <artifactId>certificate-management-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

jmx-monitoring

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>jmx-monitoring</artifactId>
    <version>7.0.0</version>
</dependency>
```

jmx-monitoring-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>jmx-monitoring-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

jmxremote_optional-repackaged

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>jmxremote_optional-repackaged</artifactId>
    <version>7.0.0</version>
</dependency>
```

management-api

Version: 3.3.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.external</groupId>
    <artifactId>management-api</artifactId>
    <version>3.3.0.payara-p1</version>
</dependency>
```

work-management

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.connectors</groupId>
    <artifactId>work-management</artifactId>
    <version>7.0.0</version>
</dependency>
```

messaging

console-jms-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-jms-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

gf-jms-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.jms</groupId>
    <artifactId>gf-jms-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

gf-jms-injection

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.jms</groupId>
    <artifactId>gf-jms-injection</artifactId>
    <version>7.0.0</version>
```

```
</dependency>
```

jakarta.jms-api

Version: 3.1.0

```
<dependency>
    <groupId>jakarta.jms</groupId>
    <artifactId>jakarta.jms-api</artifactId>
    <version>3.1.0</version>
</dependency>
```

jms-admin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.jms</groupId>
    <artifactId>jms-admin</artifactId>
    <version>7.0.0</version>
</dependency>
```

jms-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.jms</groupId>
    <artifactId>jms-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

jms-handlers

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.jms</groupId>
    <artifactId>jms-handlers</artifactId>
    <version>7.0.0</version>
</dependency>
```

jms-notifier-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
```

```
    <artifactId>jms-notifier-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

notification-jms-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>notification-jms-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

pages

jakarta.servlet.jsp-api

Version: 4.0.0

```
<dependency>
    <groupId>jakarta.servlet.jsp</groupId>
    <artifactId>jakarta.servlet.jsp-api</artifactId>
    <version>4.0.0</version>
</dependency>
```

jakarta.servlet.jsp.jstl-api

Version: 3.0.2

```
<dependency>
    <groupId>jakarta.servlet.jsp.jstl</groupId>
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
    <version>3.0.2</version>
</dependency>
```

jersey-mvc-jsp

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext</groupId>
    <artifactId>jersey-mvc-jsp</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jspcaching-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>jspcaching-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

persistence

gf-jpa-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.persistence</groupId>
    <artifactId>gf-jpa-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

hazelcast-eclipselink-coordination

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>hazelcast-eclipselink-coordination</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta.persistence-api

Version: 3.2.0

```
<dependency>
    <groupId>jakarta.persistence</groupId>
    <artifactId>jakarta.persistence-api</artifactId>
    <version>3.2.0</version>
</dependency>
```

jpa-container

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.persistence</groupId>
    <artifactId>jpa-container</artifactId>
    <version>7.0.0</version>
</dependency>
```

org.eclipse.persistence.asm

Version: 9.9.0

```
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.asm</artifactId>
    <version>9.9.0</version>
</dependency>
```

org.eclipse.persistence.core

Version: 5.0.0-b11.payara-p1

```
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.core</artifactId>
    <version>5.0.0-b11.payara-p1</version>
</dependency>
```

org.eclipse.persistence.dbws

Version: 5.0.0-b11.payara-p1

```
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.dbws</artifactId>
    <version>5.0.0-b11.payara-p1</version>
</dependency>
```

org.eclipse.persistence.jpa

Version: 5.0.0-b11.payara-p1

```
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.jpa</artifactId>
    <version>5.0.0-b11.payara-p1</version>
</dependency>
```

org.eclipse.persistence.jpa.jpql

Version: 5.0.0-b11.payara-p1

```
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.jpa.jpql</artifactId>
    <version>5.0.0-b11.payara-p1</version>
```

```
</dependency>
```

org.eclipse.persistence.jpa.modelgen.processor

Version: 5.0.0-b11.payara-p1

```
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.jpa.modelgen.processor</artifactId>
    <version>5.0.0-b11.payara-p1</version>
</dependency>
```

org.eclipse.persistence.moxy

Version: 5.0.0-b11.payara-p1

```
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.moxy</artifactId>
    <version>5.0.0-b11.payara-p1</version>
</dependency>
```

org.eclipse.persistence.oracle

Version: 5.0.0-b11.payara-p1

```
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.oracle</artifactId>
    <version>5.0.0-b11.payara-p1</version>
</dependency>
```

persistence-common

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.persistence</groupId>
    <artifactId>persistence-common</artifactId>
    <version>7.0.0</version>
</dependency>
```

restful-webservices

jaxrs-client-tracing

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>jaxrs-client-tracing</artifactId>
    <version>7.0.0</version>
</dependency>
```

jersey-bean-validation

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext</groupId>
    <artifactId>jersey-bean-validation</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-cdi1x

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext.cdi</groupId>
    <artifactId>jersey-cdi1x</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-cdi1x-servlet

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext.cdi</groupId>
    <artifactId>jersey-cdi1x-servlet</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-cdi1x-transaction

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext.cdi</groupId>
    <artifactId>jersey-cdi1x-transaction</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-client

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.core</groupId>
    <artifactId>jersey-client</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-common

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.core</groupId>
    <artifactId>jersey-common</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-container-grizzly2-http

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-grizzly2-http</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-container-servlet

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-servlet</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-entity-filtering

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext</groupId>
    <artifactId>jersey-entity-filtering</artifactId>
    <version>4.0.0.payara-p1</version>
```

```
</dependency>
```

jersey-gf-ejb

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.containers.glassfish</groupId>
    <artifactId>jersey-gf-ejb</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-hk2

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.inject</groupId>
    <artifactId>jersey-hk2</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-media-jaxb

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-jaxb</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-media-json-binding

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-binding</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-media-json-jackson

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
```

```
    <artifactId>jersey-media-json-jackson</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-media-json-processing

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-processing</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-media-moxy

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-moxy</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-media-multipart

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-multipart</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-media-sse

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-sse</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-mp-rest-client

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext.microprofile</groupId>
    <artifactId>jersey-mp-rest-client</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-mvc

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext</groupId>
    <artifactId>jersey-mvc</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-mvc-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>jersey-mvc-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

jersey-mvc-jsp

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext</groupId>
    <artifactId>jersey-mvc-jsp</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-proxy-client

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext</groupId>
    <artifactId>jersey-proxy-client</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-server

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.core</groupId>
    <artifactId>jersey-server</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

krazo-jersey

Version: 4.0.1.payara-p1

```
<dependency>
    <groupId>org.eclipse.krazo</groupId>
    <artifactId>krazo-jersey</artifactId>
    <version>4.0.1.payara-p1</version>
</dependency>
```

security

appclient.security

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
    <artifactId>appclient.security</artifactId>
    <version>7.0.0</version>
</dependency>
```

ejb.security

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
    <artifactId>ejb.security</artifactId>
    <version>7.0.0</version>
</dependency>
```

exousia

Version: 3.0.1

```
<dependency>
    <groupId>org.glassfish.exousia</groupId>
    <artifactId>exousia</artifactId>
    <version>3.0.1</version>
```

```
</dependency>
```

gson

Version: 2.11.0

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.11.0</version>
</dependency>
```

jakarta.security.enterprise-api

Version: 4.0.0

```
<dependency>
    <groupId>jakarta.security.enterprise</groupId>
    <artifactId>jakarta.security.enterprise-api</artifactId>
    <version>4.0.0</version>
</dependency>
```

jcip-annotations

Version: 1.0-1

```
<dependency>
    <groupId>com.github.stephenc.jcip</groupId>
    <artifactId>jcip-annotations</artifactId>
    <version>1.0-1</version>
</dependency>
```

nimbus-jose-jwt

Version: 10.0.1

```
<dependency>
    <groupId>com.nimbusds</groupId>
    <artifactId>nimbus-jose-jwt</artifactId>
    <version>10.0.1</version>
</dependency>
```

security

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
```

```
    <artifactId>security</artifactId>
    <version>7.0.0</version>
</dependency>
```

security-connector-oauth2-client

Version: 4.0.0.alpha3

```
<dependency>
    <groupId>fish.payara.security.connectors</groupId>
    <artifactId>security-connector-oauth2-client</artifactId>
    <version>4.0.0.alpha3</version>
</dependency>
```

security-connector-oidc-client

Version: 4.0.0.alpha3

```
<dependency>
    <groupId>fish.payara.security.connectors</groupId>
    <artifactId>security-connector-oidc-client</artifactId>
    <version>4.0.0.alpha3</version>
</dependency>
```

security-connectors-api

Version: 4.0.0.alpha3

```
<dependency>
    <groupId>fish.payara.security.connectors</groupId>
    <artifactId>security-connectors-api</artifactId>
    <version>4.0.0.alpha3</version>
</dependency>
```

security-ee

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
    <artifactId>security-ee</artifactId>
    <version>7.0.0</version>
</dependency>
```

security-services

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
    <artifactId>security-services</artifactId>
    <version>7.0.0</version>
</dependency>
```

soteria

Version: 4.0.1.payara-p1

```
<dependency>
    <groupId>org.glassfish.soteria</groupId>
    <artifactId>soteria</artifactId>
    <version>4.0.1.payara-p1</version>
</dependency>
```

soteria.spi.bean.decorator.weld

Version: 4.0.1.payara-p1

```
<dependency>
    <groupId>org.glassfish.soteria</groupId>
    <artifactId>soteria.spi.bean.decorator.weld</artifactId>
    <version>4.0.1.payara-p1</version>
</dependency>
```

websecurity

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
    <artifactId>websecurity</artifactId>
    <version>7.0.0</version>
</dependency>
```

webservices.security

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
    <artifactId>webservices.security</artifactId>
    <version>7.0.0</version>
</dependency>
```

servlet

httpspi-servlet

Version: 4.0.3

```
<dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>httpspi-servlet</artifactId>
    <version>4.0.3</version>
</dependency>
```

jakarta.servlet-api

Version: 6.1.0

```
<dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <version>6.1.0</version>
</dependency>
```

jakarta.servlet.jsp-api

Version: 4.0.0

```
<dependency>
    <groupId>jakarta.servlet.jsp</groupId>
    <artifactId>jakarta.servlet.jsp-api</artifactId>
    <version>4.0.0</version>
</dependency>
```

jakarta.servlet.jsp.jstl-api

Version: 3.0.2

```
<dependency>
    <groupId>jakarta.servlet.jsp.jstl</groupId>
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
    <version>3.0.2</version>
</dependency>
```

jaspic-servlet-utils

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>jaspic-servlet-utils</artifactId>
    <version>7.0.0</version>
```

```
</dependency>
```

jersey-cdi1x-servlet

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext.cdi</groupId>
    <artifactId>jersey-cdi1x-servlet</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jersey-container-servlet

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-servlet</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

servlet

Version: 4.0.3

```
<dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>servlet</artifactId>
    <version>4.0.3</version>
</dependency>
```

tyrus-container-servlet

Version: 2.2.2

```
<dependency>
    <groupId>org.glassfish.tyrus</groupId>
    <artifactId>tyrus-container-servlet</artifactId>
    <version>2.2.2</version>
</dependency>
```

standard-tag-library

jakarta.servlet.jsp.jstl-api

Version: 3.0.2

```
<dependency>
    <groupId>jakarta.servlet.jsp.jstl</groupId>
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
    <version>3.0.2</version>
</dependency>
```

jstl-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>jstl-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

transaction

console-jts-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-jts-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta.transaction-api

Version: 2.0.1

```
<dependency>
    <groupId>jakarta.transaction</groupId>
    <artifactId>jakarta.transaction-api</artifactId>
    <version>2.0.1</version>
</dependency>
```

jersey-cdi1x-transaction

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext.cdi</groupId>
    <artifactId>jersey-cdi1x-transaction</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

jta

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.transaction</groupId>
    <artifactId>jta</artifactId>
    <version>7.0.0</version>
</dependency>
```

jts

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.transaction</groupId>
    <artifactId>jts</artifactId>
    <version>7.0.0</version>
</dependency>
```

transaction-internal-api

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.transaction</groupId>
    <artifactId>transaction-internal-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

weld-jta

Version: 6.0.4.final

```
<dependency>
    <groupId>org.jboss.weld.module</groupId>
    <artifactId>weld-jta</artifactId>
    <version>6.0.4.final</version>
</dependency>
```

validation

hibernate-validator

Version: 9.0.1.final.payara-p1

```
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>9.0.1.final.payara-p1</version>
```

```
</dependency>
```

hibernate-validator-cdi

Version: 9.0.1.final.payara-p1

```
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator-cdi</artifactId>
    <version>9.0.1.final.payara-p1</version>
</dependency>
```

jakarta.validation-api

Version: 3.1.1

```
<dependency>
    <groupId>jakarta.validation</groupId>
    <artifactId>jakarta.validation-api</artifactId>
    <version>3.1.1</version>
</dependency>
```

jersey-bean-validation

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.ext</groupId>
    <artifactId>jersey-bean-validation</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

webservices

httpspi-servlet

Version: 4.0.3

```
<dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>httpspi-servlet</artifactId>
    <version>4.0.3</version>
</dependency>
```

jakarta.jws-api

Version: 3.0.0

```
<dependency>
    <groupId>jakarta.jws</groupId>
    <artifactId>jakarta.jws-api</artifactId>
    <version>3.0.0</version>
</dependency>
```

jakarta.ws.rs-api

Version: 4.0.0

```
<dependency>
    <groupId>jakarta.ws.rs</groupId>
    <artifactId>jakarta.ws.rs-api</artifactId>
    <version>4.0.0</version>
</dependency>
```

jakarta.xml.bind-api

Version: 4.0.5

```
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
    <version>4.0.5</version>
</dependency>
```

jakarta.xml.soap-api

Version: 3.0.2

```
<dependency>
    <groupId>jakarta.xml.soap</groupId>
    <artifactId>jakarta.xml.soap-api</artifactId>
    <version>3.0.2</version>
</dependency>
```

jakarta.xml.ws-api

Version: 4.0.3

```
<dependency>
    <groupId>jakarta.xml.ws</groupId>
    <artifactId>jakarta.xml.ws-api</artifactId>
    <version>4.0.3</version>
</dependency>
```

jaxws-rt

Version: 4.0.3

```
<dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>jaxws-rt</artifactId>
    <version>4.0.3</version>
</dependency>
```

jaxws-tools

Version: 4.0.3

```
<dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>jaxws-tools</artifactId>
    <version>4.0.3</version>
</dependency>
```

jsr109-impl

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.webservices</groupId>
    <artifactId>jsr109-impl</artifactId>
    <version>7.0.0</version>
</dependency>
```

metro-cm-api

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>metro-cm-api</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

metro-cm-impl

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>metro-cm-impl</artifactId>
    <version>4.0.4.payara-p1</version>
```

```
</dependency>
```

metro-commons

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>metro-commons</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

metro-config-api

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>metro-config-api</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

metro-config-impl

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>metro-config-impl</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

metro-glue

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.webservices</groupId>
    <artifactId>metro-glue</artifactId>
    <version>7.0.0</version>
</dependency>
```

metro-runtime-api

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
```

```
    <artifactId>metro-runtime-api</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

metro-runtime-impl

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>metro-runtime-impl</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

opentracing-jaxws

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>opentracing-jaxws</artifactId>
    <version>7.0.0</version>
</dependency>
```

org.eclipse.persistence.dbws

Version: 5.0.0-b11.payara-p1

```
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.dbws</artifactId>
    <version>5.0.0-b11.payara-p1</version>
</dependency>
```

policy

Version: 4.0.3

```
<dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>policy</artifactId>
    <version>4.0.3</version>
</dependency>
```

rt

Version: 4.0.3

```
<dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>rt</artifactId>
    <version>4.0.3</version>
</dependency>
```

rt-fi

Version: 4.0.3

```
<dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>rt-fi</artifactId>
    <version>4.0.3</version>
</dependency>
```

saaj-impl

Version: 3.0.4

```
<dependency>
    <groupId>com.sun.xml.messaging.saaj</groupId>
    <artifactId>saaj-impl</artifactId>
    <version>3.0.4</version>
</dependency>
```

servlet

Version: 4.0.3

```
<dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>servlet</artifactId>
    <version>4.0.3</version>
</dependency>
```

soap-tcp

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.webservices</groupId>
    <artifactId>soap-tcp</artifactId>
    <version>7.0.0</version>
</dependency>
```

soaptcp-api

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>soaptcp-api</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

soaptcp-impl

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>soaptcp-impl</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

stax2-api

Version: 4.3.0

```
<dependency>
    <groupId>org.codehaus.woodstox</groupId>
    <artifactId>stax2-api</artifactId>
    <version>4.3.0</version>
</dependency>
```

streambuffer

Version: 2.1.0

```
<dependency>
    <groupId>com.sun.xml.stream.buffer</groupId>
    <artifactId>streambuffer</artifactId>
    <version>2.1.0</version>
</dependency>
```

webservices-api-osgi

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>webservices-api-osgi</artifactId>
    <version>4.0.4.payara-p1</version>
```

```
</dependency>
```

webservices-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.webservices</groupId>
    <artifactId>webservices-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

webservices-extra-jdk-packages

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>webservices-extra-jdk-packages</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

webservices-osgi

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>webservices-osgi</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

webservices.security

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
    <artifactId>webservices.security</artifactId>
    <version>7.0.0</version>
</dependency>
```

ws-mex

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
```

```
    <artifactId>ws-mex</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wscompile

Version: 4.0.3

```
<dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>wscompile</artifactId>
    <version>4.0.3</version>
</dependency>
```

wsit-api

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wsit-api</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wsit-impl

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wsit-impl</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wsmc-api

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wsmc-api</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wsmc-impl

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wsmc-impl</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wsrm-api

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wsrm-api</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wsrm-impl

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wsrm-impl</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wsrx-commons

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wsrx-commons</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wsrx-testing

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wsrx-testing</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wssx-api

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wssx-api</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wssx-impl

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wssx-impl</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wstx-api

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wstx-api</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

wstx-impl

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>wstx-impl</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

xmlfilter

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>xmlfilter</artifactId>
    <version>4.0.4.payara-p1</version>
```

```
</dependency>
```

websocket

jakarta.websocket-api

Version: 2.2.0

```
<dependency>
    <groupId>jakarta.websocket</groupId>
    <artifactId>jakarta.websocket-api</artifactId>
    <version>2.2.0</version>
</dependency>
```

jakarta.websocket-client-api

Version: 2.2.0

```
<dependency>
    <groupId>jakarta.websocket</groupId>
    <artifactId>jakarta.websocket-client-api</artifactId>
    <version>2.2.0</version>
</dependency>
```

tyrus-client

Version: 2.2.2

```
<dependency>
    <groupId>org.glassfish.tyrus</groupId>
    <artifactId>tyrus-client</artifactId>
    <version>2.2.2</version>
</dependency>
```

tyrus-container-glassfish-cdi

Version: 2.2.2

```
<dependency>
    <groupId>org.glassfish.tyrus</groupId>
    <artifactId>tyrus-container-glassfish-cdi</artifactId>
    <version>2.2.2</version>
</dependency>
```

tyrus-container-glassfish-ejb

Version: 2.2.2

```
<dependency>
    <groupId>org.glassfish.tyrus</groupId>
    <artifactId>tyrus-container-glassfish-ejb</artifactId>
    <version>2.2.2</version>
</dependency>
```

tyrus-container-grizzly-client

Version: 2.2.2

```
<dependency>
    <groupId>org.glassfish.tyrus</groupId>
    <artifactId>tyrus-container-grizzly-client</artifactId>
    <version>2.2.2</version>
</dependency>
```

tyrus-container-servlet

Version: 2.2.2

```
<dependency>
    <groupId>org.glassfish.tyrus</groupId>
    <artifactId>tyrus-container-servlet</artifactId>
    <version>2.2.2</version>
</dependency>
```

tyrus-core

Version: 2.2.2

```
<dependency>
    <groupId>org.glassfish.tyrus</groupId>
    <artifactId>tyrus-core</artifactId>
    <version>2.2.2</version>
</dependency>
```

tyrus-server

Version: 2.2.2

```
<dependency>
    <groupId>org.glassfish.tyrus</groupId>
    <artifactId>tyrus-server</artifactId>
    <version>2.2.2</version>
</dependency>
```

tyrus-spi

Version: 2.2.2

```
<dependency>
    <groupId>org.glassfish.tyrus</groupId>
    <artifactId>tyrus-spi</artifactId>
    <version>2.2.2</version>
</dependency>
```

xml-binding

codemodel

Version: 4.0.7

```
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>codemodel</artifactId>
    <version>4.0.7</version>
</dependency>
```

dtd-parser

Version: 1.5.1

```
<dependency>
    <groupId>com.sun.xml.dtd-parser</groupId>
    <artifactId>dtd-parser</artifactId>
    <version>1.5.1</version>
</dependency>
```

fastinfoset

Version: 2.1.1

```
<dependency>
    <groupId>com.sun.xml.fastinfoset</groupId>
    <artifactId>fastinfoset</artifactId>
    <version>2.1.1</version>
</dependency>
```

isorelax

Version: 20090621

```
<dependency>
    <groupId>com.sun.xml.bind.jaxb</groupId>
    <artifactId>isorelax</artifactId>
    <version>20090621</version>
```

```
</dependency>
```

istack-commons-runtime

Version: 4.2.0

```
<dependency>
    <groupId>com.sun.istack</groupId>
    <artifactId>istack-commons-runtime</artifactId>
    <version>4.2.0</version>
</dependency>
```

istack-commons-tools

Version: 4.1.2

```
<dependency>
    <groupId>com.sun.istack</groupId>
    <artifactId>istack-commons-tools</artifactId>
    <version>4.1.2</version>
</dependency>
```

jackson-dataformat-xml

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.21.2</version>
</dependency>
```

jackson-module-jakarta-xmlbind-annotations

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.module</groupId>
    <artifactId>jackson-module-jakarta-xmlbind-annotations</artifactId>
    <version>2.21.2</version>
</dependency>
```

jakarta.xml.bind-api

Version: 4.0.5

```
<dependency>
    <groupId>jakarta.xml.bind</groupId>
```

```
    <artifactId>jakarta.xml.bind-api</artifactId>
    <version>4.0.5</version>
</dependency>
```

jaxb-core

Version: 4.0.7

```
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-core</artifactId>
    <version>4.0.7</version>
</dependency>
```

jaxb-jxc

Version: 4.0.7

```
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-jxc</artifactId>
    <version>4.0.7</version>
</dependency>
```

jaxb-osgi

Version: 4.0.7

```
<dependency>
    <groupId>com.sun.xml.bind</groupId>
    <artifactId>jaxb-osgi</artifactId>
    <version>4.0.7</version>
</dependency>
```

jaxb-runtime

Version: 4.0.7

```
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
    <version>4.0.7</version>
</dependency>
```

jaxb-xjc

Version: 4.0.7

```
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-xjc</artifactId>
    <version>4.0.7</version>
</dependency>
```

jersey-media-jaxb

Version: 4.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-jaxb</artifactId>
    <version>4.0.0.payara-p1</version>
</dependency>
```

relaxng-datatype

Version: 4.0.7

```
<dependency>
    <groupId>com.sun.xml.bind.external</groupId>
    <artifactId>relaxng-datatype</artifactId>
    <version>4.0.7</version>
</dependency>
```

rngom

Version: 4.0.7

```
<dependency>
    <groupId>com.sun.xml.bind.external</groupId>
    <artifactId>rngom</artifactId>
    <version>4.0.7</version>
</dependency>
```

stax-ex

Version: 2.1.0

```
<dependency>
    <groupId>org.jvnet.staxex</groupId>
    <artifactId>stax-ex</artifactId>
    <version>2.1.0</version>
</dependency>
```

txw2

Version: 4.0.7

```
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>txw2</artifactId>
    <version>4.0.7</version>
</dependency>
```

xsom

Version: 4.0.7

```
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>xsom</artifactId>
    <version>4.0.7</version>
</dependency>
```

xml-rpc

jackson-dataformat-xml

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.21.2</version>
</dependency>
```

jackson-module-jakarta-xmlbind-annotations

Version: 2.21.2

```
<dependency>
    <groupId>com.fasterxml.jackson.module</groupId>
    <artifactId>jackson-module-jakarta-xmlbind-annotations</artifactId>
    <version>2.21.2</version>
</dependency>
```

jakarta.xml.bind-api

Version: 4.0.5

```
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
    <version>4.0.5</version>
```

```
</dependency>
```

jakarta.xml.soap-api

Version: 3.0.2

```
<dependency>
    <groupId>jakarta.xml.soap</groupId>
    <artifactId>jakarta.xml.soap-api</artifactId>
    <version>3.0.2</version>
</dependency>
```

jakarta.xml.ws-api

Version: 4.0.3

```
<dependency>
    <groupId>jakarta.xml.ws</groupId>
    <artifactId>jakarta.xml.ws-api</artifactId>
    <version>4.0.3</version>
</dependency>
```

metro-xmlsec-repackaged

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>metro-xmlsec-repackaged</artifactId>
    <version>7.0.0</version>
</dependency>
```

xmlfilter

Version: 4.0.4.payara-p1

```
<dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>xmlfilter</artifactId>
    <version>4.0.4.payara-p1</version>
</dependency>
```

xmlsec

Version: 4.0.4

```
<dependency>
    <groupId>org.apache.santuario</groupId>
```

```
    <artifactId>xmlsec</artifactId>
    <version>4.0.4</version>
</dependency>
```

MicroProfile Specification Dependencies Mapping

This section details all dependencies related to Eclipse MicroProfile.

TIP	Overlapping dependencies with Jakarta EE can be visualized in the "corresponding page".

`Common`

microprofile-common

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-common</artifactId>
    <version>7.0.0</version>
</dependency>
```

`Config`

microprofile-config

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-config</artifactId>
    <version>7.0.0</version>
</dependency>
```

microprofile-config-api

Version: 3.1.1

```
<dependency>
    <groupId>org.eclipse.microprofile.config</groupId>
    <artifactId>microprofile-config-api</artifactId>
    <version>3.1.1</version>
</dependency>
```

microprofile-config-extensions

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-config-extensions</artifactId>
    <version>7.0.0</version>
</dependency>
```

microprofile-config-service

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>microprofile-config-service</artifactId>
    <version>7.0.0</version>
</dependency>
```

Fault-Tolerance

microprofile-fault-tolerance

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-fault-tolerance</artifactId>
    <version>7.0.0</version>
</dependency>
```

microprofile-fault-tolerance-api

Version: 4.0.2.payara-p1

```
<dependency>
    <groupId>org.eclipse.microprofile.fault-tolerance</groupId>
    <artifactId>microprofile-fault-tolerance-api</artifactId>
    <version>4.0.2.payara-p1</version>
</dependency>
```

Health

microprofile-health-api

Version: 4.0.1

```
<dependency>
    <groupId>org.eclipse.microprofile.health</groupId>
    <artifactId>microprofile-health-api</artifactId>
    <version>4.0.1</version>
</dependency>
```

microprofile-healthcheck

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-healthcheck</artifactId>
    <version>7.0.0</version>
</dependency>
```

JWT-Authentication

gson

Version: 2.13.1

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.13.1</version>
</dependency>
```

jcip-annotations

Version: 1.0-1

```
<dependency>
    <groupId>com.github.stephenc.jcip</groupId>
    <artifactId>jcip-annotations</artifactId>
    <version>1.0-1</version>
</dependency>
```

microprofile-jwt-auth

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-jwt-auth</artifactId>
    <version>7.0.0</version>
</dependency>
```

microprofile-jwt-auth-api

Version: 2.1

```
<dependency>
    <groupId>org.eclipse.microprofile.jwt</groupId>
    <artifactId>microprofile-jwt-auth-api</artifactId>
    <version>2.1</version>
```

```
</dependency>
```

nimbus-jose-jwt

Version: 10.9

```
<dependency>
    <groupId>com.nimbusds</groupId>
    <artifactId>nimbus-jose-jwt</artifactId>
    <version>10.9</version>
</dependency>
```

Metrics

microprofile-metrics

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-metrics</artifactId>
    <version>7.0.0</version>
</dependency>
```

microprofile-metrics-api

Version: 5.1.2

```
<dependency>
    <groupId>org.eclipse.microprofile.metrics</groupId>
    <artifactId>microprofile-metrics-api</artifactId>
    <version>5.1.2</version>
</dependency>
```

OpenAPI

microprofile-openapi

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-openapi</artifactId>
    <version>7.0.0</version>
</dependency>
```

microprofile-openapi-api

Version: 3.1.2

```
<dependency>
    <groupId>org.eclipse.microprofile.openapi</groupId>
    <artifactId>microprofile-openapi-api</artifactId>
    <version>3.1.2</version>
</dependency>
```

Rest-Client

microprofile-rest-client

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-rest-client</artifactId>
    <version>7.0.0</version>
</dependency>
```

microprofile-rest-client-api

Version: 3.0.1

```
<dependency>
    <groupId>org.eclipse.microprofile.rest.client</groupId>
    <artifactId>microprofile-rest-client-api</artifactId>
    <version>3.0.1</version>
</dependency>
```

Telemetry

ejb-opentracing

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.ejb</groupId>
    <artifactId>ejb-opentracing</artifactId>
    <version>7.0.0</version>
</dependency>
```

jctools-core

Version: 4.0.1

```
<dependency>
    <groupId>org.jctools</groupId>
    <artifactId>jctools-core</artifactId>
    <version>4.0.1</version>
</dependency>
```

microprofile-opentracing

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-opentracing</artifactId>
    <version>7.0.0</version>
</dependency>
```

microprofile-opentracing-api

Version: 3.0

```
<dependency>
    <groupId>org.eclipse.microprofile.opentracing</groupId>
    <artifactId>microprofile-opentracing-api</artifactId>
    <version>3.0</version>
</dependency>
```

microprofile-telemetry

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-telemetry</artifactId>
    <version>7.0.0</version>
</dependency>
```

opentelemetry-repackaged

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>opentelemetry-repackaged</artifactId>
    <version>7.0.0</version>
</dependency>
```

opentracing-adapter

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>opentracing-adapter</artifactId>
    <version>7.0.0</version>
```

```
</dependency>
```

opentracing-jaxws

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>opentracing-jaxws</artifactId>
    <version>7.0.0</version>
</dependency>
```

opentracing-repackaged

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>opentracing-repackaged</artifactId>
    <version>7.0.0</version>
</dependency>
```

Payara Platform Internal Dependencies

This section details all dependencies related to internal modules, extensions, plugins and other extras.

acc-config

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.appclient</groupId>
    <artifactId>acc-config</artifactId>
    <version>7.0.0</version>
</dependency>
```

accessors-smart

Version: 2.6.0

```
<dependency>
    <groupId>net.minidev</groupId>
    <artifactId>accessors-smart</artifactId>
    <version>2.6.0</version>
</dependency>
```

admin-cli

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admin</groupId>
    <artifactId>admin-cli</artifactId>
    <version>7.0.0</version>
</dependency>
```

admin-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admin</groupId>
    <artifactId>admin-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

admin-util

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admin</groupId>
    <artifactId>admin-util</artifactId>
    <version>7.0.0</version>
</dependency>
```

amx-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>amx-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

amx-javaee

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>amx-javaee</artifactId>
    <version>7.0.0</version>
```

```
</dependency>
```

ant

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>ant</artifactId>
    <version>7.0.0</version>
</dependency>
```

aopalliance-repackaged

Version: 4.0.0-m3.payara-p2

```
<dependency>
    <groupId>org.glassfish.hk2.external</groupId>
    <artifactId>aopalliance-repackaged</artifactId>
    <version>4.0.0-m3.payara-p2</version>
</dependency>
```

api-exporter

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.core</groupId>
    <artifactId>api-exporter</artifactId>
    <version>7.0.0</version>
</dependency>
```

appclient-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.appclient.server</groupId>
    <artifactId>appclient-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

appclient-server-core

Version: 7.0.0

```
<dependency>
```

```
    <groupId>fish.payara.server.internal.appclient.server</groupId>
    <artifactId>appclient-server-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

asadmin-audit

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>asadmin-audit</artifactId>
    <version>7.0.0</version>
</dependency>
```

asadmin-recorder

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>asadmin-recorder</artifactId>
    <version>7.0.0</version>
</dependency>
```

backup

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admin</groupId>
    <artifactId>backup</artifactId>
    <version>7.0.0</version>
</dependency>
```

bcel

Version: 6.12.0

```
<dependency>
    <groupId>org.apache.bcel</groupId>
    <artifactId>bcel</artifactId>
    <version>6.12.0</version>
</dependency>
```

class-model

Version: 4.0.0-m3.payara-p2

```
<dependency>
    <groupId>org.glassfish.hk2</groupId>
    <artifactId>class-model</artifactId>
    <version>4.0.0-m3.payara-p2</version>
</dependency>
```

classgraph

Version: 4.8.158

```
<dependency>
    <groupId>io.github.classgraph</groupId>
    <artifactId>classgraph</artifactId>
    <version>4.8.158</version>
</dependency>
```

classmate

Version: 1.7.3

```
<dependency>
    <groupId>com.fasterxml</groupId>
    <artifactId>classmate</artifactId>
    <version>1.7.3</version>
</dependency>
```

cluster-admin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.cluster</groupId>
    <artifactId>cluster-admin</artifactId>
    <version>7.0.0</version>
</dependency>
```

cluster-common

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.cluster</groupId>
    <artifactId>cluster-common</artifactId>
    <version>7.0.0</version>
</dependency>
```

cluster-ssh

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.cluster</groupId>
    <artifactId>cluster-ssh</artifactId>
    <version>7.0.0</version>
</dependency>
```

common-util

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>common-util</artifactId>
    <version>7.0.0</version>
</dependency>
```

concurro

Version: 3.1.0

```
<dependency>
    <groupId>org.glassfish.concurro</groupId>
    <artifactId>concurro</artifactId>
    <version>3.1.0</version>
</dependency>
```

config-api

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admin</groupId>
    <artifactId>config-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

config-types

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.hk2</groupId>
    <artifactId>config-types</artifactId>
    <version>7.0.0</version>
```

```
</dependency>
```

console-cluster-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-cluster-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

console-common

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-common</artifactId>
    <version>7.0.0</version>
</dependency>
```

console-common-full-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-common-full-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

console-corba-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-corba-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

console-jca-plugin

Version: 7.0.0

```
<dependency>
```

```
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-jca-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

console-jdbc-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-jdbc-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

console-payara-enterprise-branding-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-payara-enterprise-branding-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

console-plugin-service

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-plugin-service</artifactId>
    <version>7.0.0</version>
</dependency>
```

console-reference-manual-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-reference-manual-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

console-web-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>console-web-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

container-common

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>container-common</artifactId>
    <version>7.0.0</version>
</dependency>
```

cpu-core-reporting

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>cpu-core-reporting</artifactId>
    <version>7.0.0</version>
</dependency>
```

crac

Version: 1.5.0

```
<dependency>
    <groupId>org.crac</groupId>
    <artifactId>crac</artifactId>
    <version>1.5.0</version>
</dependency>
```

discord-notifier-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>discord-notifier-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

discord-notifier-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>discord-notifier-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

docker

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.docker</groupId>
    <artifactId>docker</artifactId>
    <version>7.0.0</version>
</dependency>
```

dol

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.deployment</groupId>
    <artifactId>dol</artifactId>
    <version>7.0.0</version>
</dependency>
```

environment-warning

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>environment-warning</artifactId>
    <version>7.0.0</version>
</dependency>
```

epicyro

Version: 3.1.1

```
<dependency>
    <groupId>org.glassfish.epicyro</groupId>
    <artifactId>epicyro</artifactId>
    <version>3.1.1</version>
```

```
</dependency>
```

eventbus-notifier-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>eventbus-notifier-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

flashlight-extra-jdk-packages

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.flashlight</groupId>
    <artifactId>flashlight-extra-jdk-packages</artifactId>
    <version>7.0.0</version>
</dependency>
```

flashlight-framework

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.flashlight</groupId>
    <artifactId>flashlight-framework</artifactId>
    <version>7.0.0</version>
</dependency>
```

gf-admingui-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>gf-admingui-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

gf-client-module

Version: 7.0.0

```
<dependency>
```

```
    <groupId>fish.payara.server.appclient</groupId>
    <artifactId>gf-client-module</artifactId>
    <version>7.0.0</version>
</dependency>
```

gf-restadmin-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admin</groupId>
    <artifactId>gf-restadmin-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

gf-web-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>gf-web-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

glassfish

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.core</groupId>
    <artifactId>glassfish</artifactId>
    <version>7.0.0</version>
</dependency>
```

glassfish-api

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>glassfish-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

glassfish-corba-csiv2-idl

Version: 5.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.corba</groupId>
    <artifactId>glassfish-corba-csiv2-idl</artifactId>
    <version>5.0.0.payara-p1</version>
</dependency>
```

glassfish-corba-internal-api

Version: 5.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.corba</groupId>
    <artifactId>glassfish-corba-internal-api</artifactId>
    <version>5.0.0.payara-p1</version>
</dependency>
```

glassfish-corba-omgapi

Version: 5.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.corba</groupId>
    <artifactId>glassfish-corba-omgapi</artifactId>
    <version>5.0.0.payara-p1</version>
</dependency>
```

glassfish-corba-orb

Version: 5.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.corba</groupId>
    <artifactId>glassfish-corba-orb</artifactId>
    <version>5.0.0.payara-p1</version>
</dependency>
```

glassfish-ee-api

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>glassfish-ee-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

glassfish-extra-jre-packages

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.core</groupId>
    <artifactId>glassfish-extra-jre-packages</artifactId>
    <version>7.0.0</version>
</dependency>
```

glassfish-grizzly-extra-all

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.grizzly</groupId>
    <artifactId>glassfish-grizzly-extra-all</artifactId>
    <version>7.0.0</version>
</dependency>
```

glassfish-javassist-packages

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.core</groupId>
    <artifactId>glassfish-javassist-packages</artifactId>
    <version>7.0.0</version>
</dependency>
```

glassfish-mbeanserver

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>glassfish-mbeanserver</artifactId>
    <version>7.0.0</version>
</dependency>
```

glassfish-naming

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>glassfish-naming</artifactId>
    <version>7.0.0</version>
```

```
</dependency>
```

glassfish-oracle-jdbc-driver-packages

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.persistence</groupId>
    <artifactId>glassfish-oracle-jdbc-driver-packages</artifactId>
    <version>7.0.0</version>
</dependency>
```

gmbal

Version: 4.1.1

```
<dependency>
    <groupId>org.glassfish.gmbal</groupId>
    <artifactId>gmbal</artifactId>
    <version>4.1.1</version>
</dependency>
```

gmbal-api-only

Version: 4.1.1

```
<dependency>
    <groupId>org.glassfish.gmbal</groupId>
    <artifactId>gmbal-api-only</artifactId>
    <version>4.1.1</version>
</dependency>
```

grizzly-npn-osgi

Version: 2.0.0

```
<dependency>
    <groupId>org.glassfish.grizzly</groupId>
    <artifactId>grizzly-npn-osgi</artifactId>
    <version>2.0.0</version>
</dependency>
```

ha-api

Version: 3.1.13

```
<dependency>
```

```
    <groupId>org.glassfish.ha</groupId>
    <artifactId>ha-api</artifactId>
    <version>3.1.13</version>
</dependency>
```

ha-file-store

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.ha</groupId>
    <artifactId>ha-file-store</artifactId>
    <version>7.0.0</version>
</dependency>
```

ha-hazelcast-store

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.ha</groupId>
    <artifactId>ha-hazelcast-store</artifactId>
    <version>7.0.0</version>
</dependency>
```

hazelcast

Version: 5.3.8-payara-1

```
<dependency>
    <groupId>com.hazelcast</groupId>
    <artifactId>hazelcast</artifactId>
    <version>5.3.8-payara-1</version>
</dependency>
```

hazelcast-bootstrap

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>hazelcast-bootstrap</artifactId>
    <version>7.0.0</version>
</dependency>
```

hazelcast-tpc-engine

Version: 5.3.8-payara-1

```
<dependency>
    <groupId>com.hazelcast</groupId>
    <artifactId>hazelcast-tpc-engine</artifactId>
    <version>5.3.8-payara-1</version>
</dependency>
```

healthcheck-checker

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>healthcheck-checker</artifactId>
    <version>7.0.0</version>
</dependency>
```

healthcheck-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>healthcheck-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

healthcheck-cpool

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>healthcheck-cpool</artifactId>
    <version>7.0.0</version>
</dependency>
```

healthcheck-metrics

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>healthcheck-metrics</artifactId>
    <version>7.0.0</version>
</dependency>
```

healthcheck-service-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>healthcheck-service-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

healthcheck-stuck

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>healthcheck-stuck</artifactId>
    <version>7.0.0</version>
</dependency>
```

hikaricp

Version: 4.0.3

```
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>hikaricp</artifactId>
    <version>4.0.3</version>
</dependency>
```

hk2

Version: 4.0.0-m3.payara-p2

```
<dependency>
    <groupId>org.glassfish.hk2</groupId>
    <artifactId>hk2</artifactId>
    <version>4.0.0-m3.payara-p2</version>
</dependency>
```

hk2-api

Version: 4.0.0-m3.payara-p2

```
<dependency>
    <groupId>org.glassfish.hk2</groupId>
    <artifactId>hk2-api</artifactId>
    <version>4.0.0-m3.payara-p2</version>
```

```
</dependency>
```

hk2-config

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.hk2</groupId>
    <artifactId>hk2-config</artifactId>
    <version>7.0.0</version>
</dependency>
```

hk2-core

Version: 4.0.0-m3.payara-p2

```
<dependency>
    <groupId>org.glassfish.hk2</groupId>
    <artifactId>hk2-core</artifactId>
    <version>4.0.0-m3.payara-p2</version>
</dependency>
```

hk2-extras

Version: 4.0.0-m3.payara-p2

```
<dependency>
    <groupId>org.glassfish.hk2</groupId>
    <artifactId>hk2-extras</artifactId>
    <version>4.0.0-m3.payara-p2</version>
</dependency>
```

hk2-locator

Version: 4.0.0-m3.payara-p2

```
<dependency>
    <groupId>org.glassfish.hk2</groupId>
    <artifactId>hk2-locator</artifactId>
    <version>4.0.0-m3.payara-p2</version>
</dependency>
```

hk2-runlevel

Version: 4.0.0-m3.payara-p2

```
<dependency>
```

```
    <groupId>org.glassfish.hk2</groupId>
    <artifactId>hk2-runlevel</artifactId>
    <version>4.0.0-m3.payara-p2</version>
</dependency>
```

hk2-utils

Version: 4.0.0-m3.payara-p2

```
<dependency>
    <groupId>org.glassfish.hk2</groupId>
    <artifactId>hk2-utils</artifactId>
    <version>4.0.0-m3.payara-p2</version>
</dependency>
```

internal-api

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>internal-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta-ee11-shim

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>jakarta-ee11-shim</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta-ee9-shim

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>jakarta-ee9-shim</artifactId>
    <version>7.0.0</version>
</dependency>
```

jakarta.mvc-api

Version: 3.0.0

```
<dependency>
    <groupId>jakarta.mvc</groupId>
    <artifactId>jakarta.mvc-api</artifactId>
    <version>3.0.0</version>
</dependency>
```

jansi-core

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jansi-core</artifactId>
    <version>3.30.11</version>
</dependency>
```

javassist

Version: 3.31.0-ga

```
<dependency>
    <groupId>org.javassist</groupId>
    <artifactId>javassist</artifactId>
    <version>3.31.0-ga</version>
</dependency>
```

jboss-classfilewriter

Version: 1.3.1.final

```
<dependency>
    <groupId>org.jboss.classfilewriter</groupId>
    <artifactId>jboss-classfilewriter</artifactId>
    <version>1.3.1.final</version>
</dependency>
```

jboss-logging

Version: 3.6.3.final

```
<dependency>
    <groupId>org.jboss.logging</groupId>
    <artifactId>jboss-logging</artifactId>
    <version>3.6.3.final</version>
</dependency>
```

jdbc-admin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.jdbc</groupId>
    <artifactId>jdbc-admin</artifactId>
    <version>7.0.0</version>
</dependency>
```

jdbc-config

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.jdbc</groupId>
    <artifactId>jdbc-config</artifactId>
    <version>7.0.0</version>
</dependency>
```

jdbc-runtime

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.jdbc</groupId>
    <artifactId>jdbc-runtime</artifactId>
    <version>7.0.0</version>
</dependency>
```

jline

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jline</artifactId>
    <version>3.30.11</version>
</dependency>
```

jline-builtins

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jline-builtins</artifactId>
    <version>3.30.11</version>
```

```
</dependency>
```

jline-native

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jline-native</artifactId>
    <version>3.30.11</version>
</dependency>
```

jline-reader

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jline-reader</artifactId>
    <version>3.30.11</version>
</dependency>
```

jline-remote-ssh

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jline-remote-ssh</artifactId>
    <version>3.30.11</version>
</dependency>
```

jline-remote-telnet

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jline-remote-telnet</artifactId>
    <version>3.30.11</version>
</dependency>
```

jline-style

Version: 3.30.11

```
<dependency>
```

```
    <groupId>org.jline</groupId>
    <artifactId>jline-style</artifactId>
    <version>3.30.11</version>
</dependency>
```

jline-terminal

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jline-terminal</artifactId>
    <version>3.30.11</version>
</dependency>
```

jline-terminal-ffm

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jline-terminal-ffm</artifactId>
    <version>3.30.11</version>
</dependency>
```

jline-terminal-jansi

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jline-terminal-jansi</artifactId>
    <version>3.30.11</version>
</dependency>
```

jline-terminal-jna

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jline-terminal-jna</artifactId>
    <version>3.30.11</version>
</dependency>
```

jline-terminal-jni

Version: 3.30.11

```
<dependency>
    <groupId>org.jline</groupId>
    <artifactId>jline-terminal-jni</artifactId>
    <version>3.30.11</version>
</dependency>
```

jsr107-repackaged

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>jsr107-repackaged</artifactId>
    <version>7.0.0</version>
</dependency>
```

kernel

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.core</groupId>
    <artifactId>kernel</artifactId>
    <version>7.0.0</version>
</dependency>
```

krazo-core

Version: 4.0.1.payara-p1

```
<dependency>
    <groupId>org.eclipse.krazo</groupId>
    <artifactId>krazo-core</artifactId>
    <version>4.0.1.payara-p1</version>
</dependency>
```

launcher

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admin</groupId>
    <artifactId>launcher</artifactId>
    <version>7.0.0</version>
</dependency>
```

ldapbp-repackaged

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>ldapbp-repackaged</artifactId>
    <version>7.0.0</version>
</dependency>
```

libpam4j

Version: 1.11

```
<dependency>
    <groupId>org.kohsuke</groupId>
    <artifactId>libpam4j</artifactId>
    <version>1.11</version>
</dependency>
```

libpam4j-repackaged

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>libpam4j-repackaged</artifactId>
    <version>7.0.0</version>
</dependency>
```

logging

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.core</groupId>
    <artifactId>logging</artifactId>
    <version>7.0.0</version>
</dependency>
```

microprofile-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>microprofile-connector</artifactId>
    <version>7.0.0</version>
```

```
</dependency>
```

microprofile-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>microprofile-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

mimepull

Version: 1.10.0

```
<dependency>
    <groupId>org.jvnet.mimepull</groupId>
    <artifactId>mimepull</artifactId>
    <version>1.10.0</version>
</dependency>
```

monitoring-console-api

Version: 3.0.0

```
<dependency>
    <groupId>fish.payara.monitoring-console</groupId>
    <artifactId>monitoring-console-api</artifactId>
    <version>3.0.0</version>
</dependency>
```

monitoring-console-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.monitoring-console</groupId>
    <artifactId>monitoring-console-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

monitoring-console-process

Version: 3.0.0

```
<dependency>
```

```
    <groupId>fish.payara.monitoring-console</groupId>
    <artifactId>monitoring-console-process</artifactId>
    <version>3.0.0</version>
</dependency>
```

monitoring-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admin</groupId>
    <artifactId>monitoring-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

newrelic-notifier-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>newrelic-notifier-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

newrelic-notifier-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>newrelic-notifier-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

notification-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>notification-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

notification-eventbus-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>notification-eventbus-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

nucleus-grizzly-all

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.grizzly</groupId>
    <artifactId>nucleus-grizzly-all</artifactId>
    <version>7.0.0</version>
</dependency>
```

nucleus-resources

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.resourcebase.resources</groupId>
    <artifactId>nucleus-resources</artifactId>
    <version>7.0.0</version>
</dependency>
```

orb-connector

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.orb</groupId>
    <artifactId>orb-connector</artifactId>
    <version>7.0.0</version>
</dependency>
```

orb-enabler

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.orb</groupId>
    <artifactId>orb-enabler</artifactId>
    <version>7.0.0</version>
</dependency>
```

orb-iiop

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.orb</groupId>
    <artifactId>orb-iiop</artifactId>
    <version>7.0.0</version>
</dependency>
```

org.apache.felix.bundlerepository

Version: 2.0.10

```
<dependency>
    <groupId>org.apache.felix</groupId>
    <artifactId>org.apache.felix.bundlerepository</artifactId>
    <version>2.0.10</version>
</dependency>
```

org.osgi.service.component

Version: 1.5.1

```
<dependency>
    <groupId>org.osgi</groupId>
    <artifactId>org.osgi.service.component</artifactId>
    <version>1.5.1</version>
</dependency>
```

org.osgi.util.function

Version: 1.2.0

```
<dependency>
    <groupId>org.osgi</groupId>
    <artifactId>org.osgi.util.function</artifactId>
    <version>1.2.0</version>
</dependency>
```

org.osgi.util.promise

Version: 1.3.0

```
<dependency>
    <groupId>org.osgi</groupId>
    <artifactId>org.osgi.util.promise</artifactId>
    <version>1.3.0</version>
```

```
</dependency>
```

osgi-adapter

Version: 4.0.0-m3.payara-p2

```
<dependency>
    <groupId>org.glassfish.hk2</groupId>
    <artifactId>osgi-adapter</artifactId>
    <version>4.0.0-m3.payara-p2</version>
</dependency>
```

osgi-cli-remote

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.osgi-platforms</groupId>
    <artifactId>osgi-cli-remote</artifactId>
    <version>7.0.0</version>
</dependency>
```

osgi-container

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.osgi-platforms</groupId>
    <artifactId>osgi-container</artifactId>
    <version>7.0.0</version>
</dependency>
```

osgi-resource-locator

Version: 3.0.0

```
<dependency>
    <groupId>org.glassfish.hk2</groupId>
    <artifactId>osgi-resource-locator</artifactId>
    <version>3.0.0</version>
</dependency>
```

payara-api

Version: 7.0.0

```
<dependency>
```

```
    <groupId>fish.payara.api</groupId>
    <artifactId>payara-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

payara-console-extras

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>payara-console-extras</artifactId>
    <version>7.0.0</version>
</dependency>
```

payara-executor-service

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>payara-executor-service</artifactId>
    <version>7.0.0</version>
</dependency>
```

payara-insight-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>payara-insight-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

payara-jsr107

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>payara-jsr107</artifactId>
    <version>7.0.0</version>
</dependency>
```

payara-micro-boot

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.extras</groupId>
    <artifactId>payara-micro-boot</artifactId>
    <version>7.0.0</version>
</dependency>
```

payara-micro-service

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>payara-micro-service</artifactId>
    <version>7.0.0</version>
</dependency>
```

payara-rest-endpoints

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>payara-rest-endpoints</artifactId>
    <version>7.0.0</version>
</dependency>
```

pfl-basic

Version: 5.1.0

```
<dependency>
    <groupId>org.glassfish.pfl</groupId>
    <artifactId>pfl-basic</artifactId>
    <version>5.1.0</version>
</dependency>
```

pfl-basic-tools

Version: 5.1.0

```
<dependency>
    <groupId>org.glassfish.pfl</groupId>
    <artifactId>pfl-basic-tools</artifactId>
    <version>5.1.0</version>
</dependency>
```

pfl-dynamic

Version: 5.1.0

```
<dependency>
    <groupId>org.glassfish.pfl</groupId>
    <artifactId>pfl-dynamic</artifactId>
    <version>5.1.0</version>
</dependency>
```

pfl-tf

Version: 5.1.0

```
<dependency>
    <groupId>org.glassfish.pfl</groupId>
    <artifactId>pfl-tf</artifactId>
    <version>5.1.0</version>
</dependency>
```

pfl-tf-tools

Version: 5.1.0

```
<dependency>
    <groupId>org.glassfish.pfl</groupId>
    <artifactId>pfl-tf-tools</artifactId>
    <version>5.1.0</version>
</dependency>
```

phonehome-bootstrap

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>phonehome-bootstrap</artifactId>
    <version>7.0.0</version>
</dependency>
```

realm-stores

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
    <artifactId>realm-stores</artifactId>
    <version>7.0.0</version>
```

```
</dependency>
```

requesttracing-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-modules</groupId>
    <artifactId>requesttracing-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

rest-client

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admin</groupId>
    <artifactId>rest-client</artifactId>
    <version>7.0.0</version>
</dependency>
```

rest-client-ssl

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>rest-client-ssl</artifactId>
    <version>7.0.0</version>
</dependency>
```

rest-monitoring-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admingui</groupId>
    <artifactId>rest-monitoring-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

rest-monitoring-service

Version: 7.0.0

```
<dependency>
```

```
    <groupId>fish.payara.server.internal.payara-appserver-modules</groupId>
    <artifactId>rest-monitoring-service</artifactId>
    <version>7.0.0</version>
</dependency>
```

rest-service

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.admin</groupId>
    <artifactId>rest-service</artifactId>
    <version>7.0.0</version>
</dependency>
```

rmic

Version: 5.0.0.payara-p1

```
<dependency>
    <groupId>org.glassfish.corba</groupId>
    <artifactId>rmic</artifactId>
    <version>5.0.0.payara-p1</version>
</dependency>
```

scattered-archive-api

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>scattered-archive-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

simple-glassfish-api

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>simple-glassfish-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

slack-notifier-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>slack-notifier-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

slack-notifier-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>slack-notifier-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

snakeyaml

Version: 2.6

```
<dependency>
    <groupId>org.yaml</groupId>
    <artifactId>snakeyaml</artifactId>
    <version>2.6</version>
</dependency>
```

snakeyaml-engine

Version: 2.6

```
<dependency>
    <groupId>org.snakeyaml</groupId>
    <artifactId>snakeyaml-engine</artifactId>
    <version>2.6</version>
</dependency>
```

snmp-notifier-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>snmp-notifier-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

snmp-notifier-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>snmp-notifier-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

snmp4j

Version: 3.8.2

```
<dependency>
    <groupId>org.snmp4j</groupId>
    <artifactId>snmp4j</artifactId>
    <version>3.8.2</version>
</dependency>
```

snmp4j-repackaged

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>snmp4j-repackaged</artifactId>
    <version>7.0.0</version>
</dependency>
```

ssl-impl

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.security</groupId>
    <artifactId>ssl-impl</artifactId>
    <version>7.0.0</version>
</dependency>
```

stats77

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.common</groupId>
    <artifactId>stats77</artifactId>
    <version>7.0.0</version>
```

```
</dependency>
```

teams-notifier-console-plugin

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>teams-notifier-console-plugin</artifactId>
    <version>7.0.0</version>
</dependency>
```

teams-notifier-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.extensions.notifiers</groupId>
    <artifactId>teams-notifier-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

tiger-types

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>tiger-types</artifactId>
    <version>7.0.0</version>
</dependency>
```

trilead-ssh2

Version: build-217-jenkins-16

```
<dependency>
    <groupId>org.jenkins-ci</groupId>
    <artifactId>trilead-ssh2</artifactId>
    <version>build-217-jenkins-16</version>
</dependency>
```

trilead-ssh2-repackaged

Version: 7.0.0

```
<dependency>
```

```
    <groupId>fish.payara.server.internal.packager</groupId>
    <artifactId>trilead-ssh2-repackaged</artifactId>
    <version>7.0.0</version>
</dependency>
```

war-util

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>war-util</artifactId>
    <version>7.0.0</version>
</dependency>
```

wasp

Version: 4.0.0

```
<dependency>
    <groupId>org.glassfish.wasp</groupId>
    <artifactId>wasp</artifactId>
    <version>4.0.0</version>
</dependency>
```

web-cli

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>web-cli</artifactId>
    <version>7.0.0</version>
</dependency>
```

web-core

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>web-core</artifactId>
    <version>7.0.0</version>
</dependency>
```

web-embed-api

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>web-embed-api</artifactId>
    <version>7.0.0</version>
</dependency>
```

web-glue

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>web-glue</artifactId>
    <version>7.0.0</version>
</dependency>
```

web-gui-plugin-common

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>web-gui-plugin-common</artifactId>
    <version>7.0.0</version>
</dependency>
```

web-ha

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>web-ha</artifactId>
    <version>7.0.0</version>
</dependency>
```

web-naming

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>web-naming</artifactId>
    <version>7.0.0</version>
</dependency>
```

web-sse

Version: 7.0.0

```
<dependency>
    <groupId>fish.payara.server.internal.web</groupId>
    <artifactId>web-sse</artifactId>
    <version>7.0.0</version>
</dependency>
```

woodstox-core

Version: 7.1.1

```
<dependency>
    <groupId>com.fasterxml.woodstox</groupId>
    <artifactId>woodstox-core</artifactId>
    <version>7.1.1</version>
</dependency>
```

xsdlib

Version: 2022.7

```
<dependency>
    <groupId>net.java.dev.msv</groupId>
    <artifactId>xsdlib</artifactId>
    <version>2022.7</version>
</dependency>
```

Release Notes

Release Notes

Detailed information about features and fixes in every release.

All Release Notes

Month	Release
2026	
July	"Release Notes 7.2.0"
June	"Release Notes 7.1.0"

Month	Release
May	"Release Notes 7.0.0"

Release Notes - Azul Payara 7.2.0

link_to_pdf:/payara/pdfs/7/azul-payara-documentation-7.2.0.pdf[Full Documentation PDF]

Supported APIs and Applications

- Jakarta EE 11
- Jakarta EE 11 Applications
- MicroProfile 6.1

Security Fixes

- [FISH-13687] Brute Force Attack Prevention

Bug Fixes

- [FISH-12353] Fix IBM MQ Resource Adapter Failing to Deploy When Using a Custom `metrics.xml` File
- [FISH-13041] [Community Contribution - lprimak] Prevent Stack Trace Caching by JVM Thread Stats Provider
- [FISH-13207] Fix `restart-deployment-group` Command Not Showing Progress
- [FISH-13878] Fix Attributes Removed from `org.glassfish.grizzly.config.dom.Http` Remaining in `httpAttr.inc`

Improvements

- [FISH-13490] Add Backwards Compatibility for Defining Implicit CDI via `glassfish-application.xml`
- [FISH-13982] Add Payara Major Versioned Deployment Descriptors

Component Upgrades

- [FISH-13426] Upgrade Grizzly to 5.0.2

- [FISH-13780] Upgrade `com.fasterxml.woodstox:woodstox-core` from 7.2.0 to 7.2.1
- [FISH-13786] Upgrade `jakarta.mvc:jakarta.mvc-api` from 3.0.0 to 3.0.1
- [FISH-13812] Upgrade `net.java.dev.jna:jna` from 5.18.1 to 5.19.1
- [FISH-13819] Upgrade `io.projectreactor:reactor-core` from 3.8.5 to 3.8.6
- [FISH-13821] Upgrade `smallrye-common.version` from 2.18.1 to 2.19.0
- [FISH-13884] Upgrade `jline.version` from 3.30.13 to 3.30.15
- [FISH-13965] Upgrade `smallrye-config.version` from 3.17.2 to 3.18.0
- [FISH-13987] Upgrade `com.fasterxml.jackson:jackson-bom` from 2.22.0 to 2.22.1

Release Notes - Azul Payara 7.1.0

link_to_pdf:/payara/pdfs/7/azul-payara-documentation-7.1.0.pdf[Full Documentation PDF]

Supported APIs and Applications

- Jakarta EE 11
- Jakarta EE 11 Applications
- MicroProfile 6.1

Security Fixes

- [FISH-13637] Fix CSRF + SSRF in Admin Console / REST Management Interface

Bug Fixes

- [FISH-10485] Fix Payara Micro Boot Properties Not Applied With the Right Timing
- [FISH-11896] Fix Quoted-Argument Parsing in `prebootCommandFile` (asadmin) and `PREBOOT_COMMANDS` (Docker)
- [FISH-12270] Fix Misleading Log Entries Generated While Configuring Additional Properties for Virtual Servers

- [FISH-12800] Fix MicroProfile `/metrics` Endpoint Containing Duplicate Metrics
- [FISH-13058] Fix Missing `JSESSIONIDVERSION` Leading to Session Not Being Correctly Retrieved in Relaxed Mode
- [FISH-13091] Assorted Deployment Group Fixes
- [FISH-13137] Fix Prometheus Metrics to Report Histogram Times in Seconds
- [FISH-13138] Fix Payara Micro's `--outputlauncher` to Respect `payara-boot.properties`
- [FISH-13400] Fix WELD-001408 Unsatisfied Dependency for Repository Injection in CDI Bean
- [FISH-13447] Fix Micro `NullPointerException` with `log-to-console` Access Log Option
- [FISH-13779] Fix Changing the Admin Password in the Admin Console Not Working

Improvements

- [FISH-11606] Avoid Parsing Method Names on Every Jakarta Data HTTP Request
- [FISH-12426] Reintroduce Standalone OpenMQ Support for Windows
- [FISH-13097] Add Default OpenMQ Credentials Warning
- [FISH-13339] Add `warlibs` Support to `remove-library` Command
- [FISH-13340] Add `warlibs` Support to `list-libraries` Command
- [FISH-13475] Support Method-Level `@Transactional` Overrides on Repository Interfaces in Jakarta Data
- [FISH-13649] Define `jakarta.security.jacc.policy.provider` for Payara Micro
- [FISH-13707] Add Alternate Parsson to BOM

Component Upgrades

- [FISH-12986] Upgrade EclipseLink from 5.0.0-B11 to 5.0.0

- [FISH-13427] Upgrade HK2 to 4.0.1
- [FISH-13477] Upgrade `commons-io:commons-io` from 2.21.0 to 2.22.0
- [FISH-13521] Upgrade `pfl.version` from 5.1.0 to 5.1.1
- [FISH-13522] Upgrade `payara.security-connectors.version` from 4.0.0.Alpha3 to 4.0.0
- [FISH-13540] Upgrade `org.glassfish:genericra` from 2.0-20081121 to 2.1-20100913
- [FISH-13568] Upgrade `com.fasterxml.jackson:jackson-bom` from 2.21.2 to 2.21.3
- [FISH-13569] Upgrade `gmbal.version` from 4.1.1 to 4.1.2
- [FISH-13609] Upgrade `jline.version` from 3.30.11 to 3.30.13
- [FISH-13667] Upgrade `org.jvnet.mimepull:mimepull` from 1.10.0 to 1.11.0
- [FISH-13669] Upgrade `asm.version` from 9.9.1 to 9.10.1
- [FISH-13704] Upgrade `jaxb-impl.version` from 4.0.7 to 4.0.9
- [FISH-13705] Upgrade `com.fasterxml.woodstox:woodstox-core` from 7.1.1 to 7.2.0
- [FISH-13710] Upgrade `jakarta.json.bind:jakarta.json.bind-api` from 3.0.1 to 3.0.2
- [FISH-13744] Upgrade `org.eclipse.persistence:org.eclipse.persistence.asm` from 9.9.0 to 9.10.0
- [FISH-13747] Upgrade `org.glassfish.jaxb:codemodel` from 4.0.7 to 4.0.9
- [FISH-13748] Upgrade Jersey to 4.0.2
- [FISH-13758] Upgrade `com.fasterxml.jackson:jackson-bom` from 2.21.3 to 2.22.0
- [FISH-13759] Upgrade `org.glassfish.grizzly:grizzly-npn-osgi` from 2.0.0

to 2.0.1

- [FISH-13760] Upgrade `com.nimbusds:nimbus-jose-jwt` from 10.9 to 10.9.1
- [FISH-13761] Upgrade `org.jetbrains.kotlin:kotlin-stdlib` from 2.3.20 to 2.4.0

Release Notes - Payara Platform Enterprise 7.0.0

link_to_pdf:/payara/pdfs/7/Payara-Enterprise-7.0.0-documentation.pdf[Full Documentation PDF]

NOTE	Payara Platform Enterprise 7.0.0 is the first Enterprise release supporting Jakarta EE 11. The minimum required Java version is JDK 21.

Supported APIs and Applications

- Jakarta EE 11
- Jakarta EE 11 Applications
- MicroProfile 6.1

Jakarta EE Certification

Jakarta EE Certification Summary

Payara Server is a certified Jakarta EE implementation. The following specifications are fully certified:

- Jakarta EE 11

As proof of compatibility, output of Jakarta EE Compatibility Test Suite tests are presented for each certified version

7.1.0

Platform TCK Results

As required by the Eclipse Foundation Technology Compatibility Kit License, following is a summary of the TCK results for Payara Server releases for Jakarta EE Platform 11.

Payara Server 7.1.0 Jakarta EE 11 Platform Certification Summary

As required by the Eclipse Foundation Technology Compatibility Kit License, following is a summary of the TCK results for Payara Server releases for Jakarta EE Platform 11.

- Product Name, Version and download URL (if applicable):

Payara Server 7.1.0

- Specification Name, Version and download URL:

  Jakarta EE Platform 11

  Jakarta EE Platform TCK 11.0.1.zip , SHA-256: `b2c0ad6db0514b75ff612dd7d855a3976c8ae9f240a3b690a684f61bf503bead`

- Public URL of TCK Results Summary:

  7.1.0 Platform TCK Results

  - Jakarta Activation 2.1.2

    See test results in section `activation`.
    Download URL & SHA-256: activation-tck-2.1.2.zip, SHA-256: `6c4aad27e45761dd9f3e0f8506f37edea41f42401465db750689145718b27a0b`

  - Jakarta Authentication 3.1.0

    See test results in section `authentication`.
    Download URL & SHA-256: jakarta-authentication-tck-3.1.0.zip, SHA-256: `8b916f1b4aed828337bd88b34bb39b133f04611c2dfe71541c2ec5d2dd22cd54`

- Jakarta Annotations 3.0.0

  See test results in section `annotations`.

  Download URL & SHA-256: jakarta-annotations-tck-3.0.0.zip, SHA-256: `8b916f1b4aed828337bd88b34bb39b133f04611c2dfe71541c2ec5d2dd22cd54`

- Jakarta Batch 2.1.5 TCK

  See test results in section `batch`.

  Download URL & SHA-256: jakarta.batch.official.tck-2.1.5.zip, SHA-256: `0dd8ca0f35cc696ea86d0dffaa1301cf2786806832ea1b2a491d528eaa57b3b7`

- Jakarta Concurrency 3.1.1 TCK

  See test results in section `concurrency`.

  Download URL & SHA-256: concurrency-tck-3.1.1.zip, SHA-256: `22728d729f620d6a85ae903e7d1184e0a7508a4328491b785f1b4f3d7215ca93`

- Jakarta Contexts and Dependency Injection 4.1.0 TCK

  See test results in sections `cdi` and `cdi-lang-model`.

  Download URL & SHA-256: cdi-tck-4.1.0-dist.zip, SHA-256: `566c547e1a9c66792eefcc6feafea87ab0c0f2e3f71385bf96865359a685df00`

- Jakarta Debugging Support for Other Languages 2.0.0 TCK

  See test results in section `debugging`.

  Download URL & SHA-256: jakarta-debugging-tck-2.0.0.zip, SHA-256: `71999815418799837dc6f3d0dc40c3dcc4144cd90c7cdfd06aa69270483d78bc`

- Jakarta Dependency Injection 2.0.2 TCK

  See test results in section `di`.

Download URL & SHA-256: jakarta.inject-tck-2.0.2-bin.zip, SHA-256: `23bce4317ca061c3de648566cdf65c74b57e1264d6891f366567955d6b834972`

- Jakarta JSON Binding 3.0.0 TCK

  See test results in section `jsonb`.

  Download URL & SHA-256: jakarta-jsonb-tck-3.0.0.zip, SHA-256: `954fd9a3a67059ddeabe5f51462a6a3b542c94fc798094dd8c312a6a28ef2d0b`

- Jakarta JSON Processing 2.1.1 TCK

  See test results in section `jsonp`.

  Download URL & SHA-256: jakarta-jsonp-tck-2.1.1.zip, SHA-256: `949f203de84deffa8c7892b555918e42f1dd220ccb7b6800741ea58af62737c1`

- Jakarta Faces 4.1.1 TCK

  See test results in section `jsf`.

  Download URL & SHA-256: jakarta-faces-tck-4.1.1.zip, SHA-256: `153184ad38e522c45e74e5da7dd01b1f3be485f56e241a72728caa43479b07ec`

- Jakarta Mail 2.1.0 TCK

  See test results in section `javamail`.

  Download URL & SHA-256: jakarta-mail-tck-2.1.0.zip, SHA-256: `6f02a92e0a5ef60260e65f95938cc566da2f93a3d269c3b321da0d787a3448a5`

- Jakarta RESTful Web Services 4.0.0 TCK

  See test results in section `jaxrs`.

  Download URL & SHA-256: jakarta-restful-ws-tck-4.0.0.zip, SHA-256: `9494caaf67a3e5524a5ee1d6e4b749782ca1cbc920c6451583b6249e51c4ac18`

- Jakarta Security 4.0.0 TCK

  See test results in section `security`.

  Download URL & SHA-256: jakarta-security-tck-4.0.0.zip, SHA-256: `e21a3d8251bf954c99a3f9a5f71911696509a1ed40c23673760785a3b88e4172`

- Jakarta Authorization 3.0.0 TCK

  See test results in section `authorization`.

  Download URL & SHA-256: jakarta-authorization-tck-3.0.0.zip, SHA-256: `70f2e15334cfc525fa3b2ab575a96bd06cde942302d45d12da040242ff969fe8`

- Jakarta Tags 3.0.1 TCK

  See test results in section `tags`.

  Download URL & SHA-256: jakarta-tags-tck-3.0.1.zip, SHA-256: `b779d125ea36480b206629830ca80c1e506f65e6e8235b22e846a9cb40aaba04`

- Jakarta Validation 3.1.1 TCK

  See test results in section `validation`.

  Download URL & SHA-256: validation-tck-dist-3.1.1.zip, SHA-256: `bda9de47960845a7c73aa9a1616e4bbed3e346de3516f768819cf5af77bf969f`

- Jakarta WebSocket 2.2.0 TCK

  See test results in section `websocket`.

  Download URL & SHA-256: jakarta-websocket-tck-2.2.0.zip, SHA-256: `ff50825305bd1318a4e6bdfab21b829cfd8cf89fab8a970f007b52636d3f2629`

- Jakarta Pages 4.0.0 TCK

  See test results in section `pages`.

Download URL & SHA-256: jakarta-pages-tck-4.0.0.zip, SHA-256: `5446aa866601a7c4c425f74054590cfc7f4fc05592a572d399ecc8694e265489`

- Jakarta Servlet 6.1.0 TCK

  See test results in section `servlet`.

  Download URL & SHA-256: jakarta-servlet-tck-6.1.0.zip, SHA-256: `1170697a87622a920bd50f0a68ed594c6efc5e17dd370566312194c4e12bec29`

- Jakarta Persistence 3.2.0 TCK

  See test results in section `persistence`.

  Download URL & SHA-256: jakarta-persistence-tck-3.2.0.zip, SHA-256: `db74101c3f52f737f57804271ea9c845cb1941f2a0685284c54c4ad4fa1f05c7`

- Jakarta Expression-Language 6.0.0 TCK

  See test results in section `expression-language`.

  Download URL & SHA-256: jakarta-expression-language-tck-6.0.0.zip, SHA-256: `90cc60b9d3340675e3224d9c4ab7a82b3cf63857f39f0006e143277eaa82bbce`

- Jakarta Data 1.0.1 TCK

  See test results in section `data`.

  Download URL & SHA-256: data-tck-1.0.1.zip, SHA-256: `92babb604b4e346f8d19929979c1d6aaf013fdf990b5e97bdd95502ad7db5b4c`

- Java runtime used to run the implementation:

  `OpenJDK Runtime Environment zulu21.0.7 (build 21.0.7+6)`

- Summary of the information for the certification environment:

Apache Derby, Linux, Ubuntu 20.04 LTS

Test Results

activation

```
Completed running 91 tests.
Number of Tests Passed      = 91
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests Not Run     = 0
Pluggability:
Completed running 2 tests.
Number of Tests Passed      = 2
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests Not Run     = 0
```

annotations

```
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

assembly

```
Completed running 12 tests.
Number of Tests Passed      = 12
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

authentication

```
Completed running 163 tests.
Number of Tests Passed      = 163
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

authorization

```
Completed running 53 tests.
Number of Tests Passed      = 53
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

batch

```
Completed running 390 tests.
Number of Tests Passed      = 372
Number of Tests Failed      = 0
```

```
Number of Tests with Errors = 0
Number of Tests skipped     = 18
```

cdi-lang-model

```
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

cdi-platform-tck

```
Completed running 1904 tests.
Number of Tests Passed      = 1904
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

cdi

```
Completed running 1334 tests.
Number of Tests Passed      = 1334
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

concurrency

```
Completed running 295 tests.
Number of Tests Passed      = 268
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 27
```

connector

```
Completed running 441 tests.
Number of Tests Passed      = 441
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

data

```
Completed running 99 tests.
Number of Tests Passed      = 99
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

dsol

```
Completed running 1 tests.
Number of Tests Passed      = 1
```

```
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

ejb30-platform

```
    assembly-tck:
    Completed running 46 tests.
    Number of Tests Passed      = 46
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    lite-tck:
    Completed running 1760 tests.
    Number of Tests Passed      = 1760
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    misc-tck:
    Completed running 98 tests.
    Number of Tests Passed      = 98
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    bb-tck:
    Completed running 1081 tests.
    Number of Tests Passed      = 1081
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    timer-tck:
    Completed running 178 tests.
    Number of Tests Passed      = 178
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    tx-tck:
    Completed running 84 tests.
    Number of Tests Passed      = 84
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    zombie-tck:
    Completed running 1 tests.
    Number of Tests Passed      = 1
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

ejb32-platform-tck

```
    lite-timer:
    Completed running 2 tests.
    Number of Tests Passed      = 2
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    lite-timer-basic:
    Completed running 40 tests.
```

```
    Number of Tests Passed      = 40
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    lite-timer-interceptor:
    Completed running 60 tests.
    Number of Tests Passed      = 60
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    lite-timer-schedule:
    Completed running 536 tests.
    Number of Tests Passed      = 536
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    lite-timer-service:
    Completed running 8 tests.
    Number of Tests Passed      = 8
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    lite-timer-timerconfig:
    Completed running 36 tests.
    Number of Tests Passed      = 36
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    mdb:
    Completed running 1 tests.
    Number of Tests Passed      = 1
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    relaxedclientview:
    Completed running 21 tests.
    Number of Tests Passed      = 21
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

expression-language-platform

```
    Completed running 717 tests.
    Number of Tests Passed      = 717
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

expression-language-tck

```
    Completed running 361 tests.
    Number of Tests Passed      = 361
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

faces-tck

```
Completed running 314 tests.
Number of Tests Passed      = 314
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 17

old-tck:
Completed running 5389 tests
Number of tests failed 0
Number of tests with errors 0
Number of tests skipped 0
```

inject-tck

```
Completed running 50 tests.
Number of Tests Passed      = 50
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

integration-tck

```
Completed running 15 tests.
Number of Tests Passed      = 15
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

javaee-module-tck

```
Completed running 24 tests.
Number of Tests Passed      = 24
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

jdbc-platform-tck

```
Completed running 4924 tests.
Number of Tests Passed      = 4924
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

jsonb-platform

```
Completed running 18 tests.
Number of Tests Passed      = 18
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

jsonb-tck

```
Completed running 295 tests.
Number of Tests Passed      = 290
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 5
```

jsonp-platform-tck

```
Completed running 76 tests.
Number of Tests Passed      = 76
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

jsonp-tck

```
glassfish-runner:
Completed running 179 tests.
Number of Tests Passed      = 179
Number of Tests Failed      = 0
Number of Tests with Errors = 0

glassfish-runner-pluggability:
Completed running 18 tests.
Number of Tests Passed      = 18
Number of Tests Failed      = 0
Number of Tests with Errors = 0

payara-runner:
Completed running 196 tests.
Number of Tests Passed      = 196
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

javamail

```
Completed running 112 tests.
Number of Tests Passed      = 112
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

messaging-platform-tck

```
Completed running 3510 tests.
Number of Tests Passed      = 3489
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 21
```

mvc-tck

```
Completed running 135 tests.
```

```
Number of Tests Passed      = 135
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

pages-platform-tck

```
Completed running 28 tests.
Number of Tests Passed      = 28
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

pages-tck

```
Completed running 682 tests.
Number of Tests Passed      = 682
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

persistence

```
core1:
Completed running 238 tests.
Number of Tests Passed      = 238
Number of Tests Failed      = 0
Number of Tests with Errors = 0

core2:
Completed running 112 tests.
Number of Tests Passed      = 112
Number of Tests Failed      = 0
Number of Tests with Errors = 0

core3:
Completed running 521 tests.
Number of Tests Passed      = 520
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 1

core4:
Completed running 405 tests.
Number of Tests Passed      = 404
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 1
```

persistence-platform-tck

```
annotations:
Completed running 984 tests.
Number of Tests Passed      = 984
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

```
    basic:
    Completed running 12 tests.
    Number of Tests Passed      = 12
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    callback:
    Completed running 396 tests.
    Number of Tests Passed      = 396
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    core-entitytest:
    Completed running 904 tests.
    Number of Tests Passed      = 894
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 10

    criteria-api:
    Completed running 3726 tests.
    Number of Tests Passed      = 3726
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    derived-id:
    Completed running 72 tests.
    Number of Tests Passed      = 72
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    ee:
    Completed running 180 tests.
    Number of Tests Passed      = 162
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 18

    entity-graph:
    Completed running 78 tests.
    Number of Tests Passed      = 78
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    entity-manager-2:
    Completed running 132 tests.
    Number of Tests Passed      = 132
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    entity-manager:
    Completed running 200 tests.
    Number of Tests Passed      = 200
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    entity-manager-factory:
    Completed running 42 tests.
    Number of Tests Passed      = 42
    Number of Tests Failed      = 0
```

```
    Number of Tests with Errors = 0

    entity-manager-factory-close-exceptions:
    Completed running 6 tests.
    Number of Tests Passed      = 6
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    entity-transaction:
    Completed running 20 tests.
    Number of Tests Passed      = 20
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    entitytest:
    Completed running 96 tests.
    Number of Tests Passed      = 96
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    enums:
    Completed running 317 tests.
    Number of Tests Passed      = 317
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    exceptions:
    Completed running 34 tests.
    Number of Tests Passed      = 34
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    inheritance:
    Completed running 60 tests.
    Number of Tests Passed      = 60
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    jpa:
    Completed running 4 tests.
    Number of Tests Passed      = 4
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    jpa22:
    Completed running 94 tests.
    Number of Tests Passed      = 94
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    lock:
    Completed running 36 tests.
    Number of Tests Passed      = 36
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    metamodel-api:
    Completed running 1560 tests.
    Number of Tests Passed      = 1560
    Number of Tests Failed      = 0
```

```
    Number of Tests with Errors = 0

    nested-embedding:
    Completed running 18 tests.
    Number of Tests Passed      = 18
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    override:
    Completed running 156 tests.
    Number of Tests Passed      = 156
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    query:
    Completed running 1242 tests.
    Number of Tests Passed      = 1242
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    relationship:
    Completed running 198 tests.
    Number of Tests Passed      = 198
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    stored-procedure-query:
    Completed running 238 tests.
    Number of Tests Passed      = 238
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    types:
    Completed running 306 tests.
    Number of Tests Passed      = 306
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    unit-util:
    Completed running 18 tests.
    Number of Tests Passed      = 18
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    util:
    Completed running 6 tests.
    Number of Tests Passed      = 6
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    versioning:
    Completed running 6 tests.
    Number of Tests Passed      = 6
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    various-tck:
    Completed running 685 tests.
    Number of Tests Passed      = 685
    Number of Tests Failed      = 0
```

```
Number of Tests with Errors = 0
```

rest

```
platform
Completed running 129 tests.
Number of Tests Passed      = 129
Number of Tests Failed      = 0
Number of Tests with Errors = 0

payara-se-tests
Completed running 7 tests.
Number of Tests Passed      = 7
Number of Tests Failed      = 0
Number of Tests with Errors = 0

payara-server-remote-tests
Completed running 2667 tests.
Number of Tests Passed      = 2607
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 60
```

security

```
Completed running 133 tests.
Number of Tests Passed      = 133
Number of Tests Failed      = 0
Number of Tests with Errors = 0

Old TCK:
Completed running 83 tests.
Number of Tests Passed      = 83
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

servlet

```
Completed running 1716 tests.
Number of Tests Passed      = 1716
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

signature

```
Completed running 4 tests.
Number of Tests Passed      = 4
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

soap

```
Completed running 447 tests.
```

```
Number of Tests Passed      = 447
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

tags

```
Completed running 541 tests.
Number of Tests Passed      = 541
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

transaction-platform-tck

```
Completed running 195 tests.
Number of Tests Passed      = 195
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

websocket-platform-tck

```
Completed running 12 tests.
Number of Tests Passed      = 12
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

websocket-tck

```
Completed running 737 tests.
Number of Tests Passed      = 737
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

xa-platform-tck

```
Completed running 66 tests.
Number of Tests Passed      = 66
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

xml-binding-tck

```
Completed running 24624 tests.
Number of Tests Passed      = 24624
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

xml-ws-tck

```
Completed running 954 tests.
Number of Tests Passed      = 954
```

```
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

Web TCK Results

As required by the Eclipse Foundation Technology Compatibility Kit License, following is a summary of the TCK results for Payara Server releases for Jakarta EE Platform 11.

Payara Server 7.1.0 (Web Profile) Jakarta EE 11 Web Profile Certification Summary

- Product Name, Version and download URL (if applicable):

Payara Server 7.1.0 (Web Profile)

- Specification Name, Version and download URL:

  Jakarta EE Web Profile 11

- TCK Version, digital SHA-256 fingerprint and download URL:

  Jakarta EE Platform TCK 11.0.1.zip , SHA-256: `b2c0ad6db0514b75ff612dd7d855a3976c8ae9f240a3b690a684f61bf503bead`

- Public URL of TCK Results Summary:

  7.1.0 Web TCK Results

- Any Additional Specification Certification Requirements:

  - Jakarta Authentication 3.1.0

    See test results in section `authentication`.

    Download URL & SHA-256: jakarta-authentication-tck-3.1.0.zip, SHA-256: `9421c6ca66274d32dfb408848f75a42d57f120599fe0d8403c5c5c1141d5ac4d`

- Jakarta Annotations 3.0.0

  See test results in section `annotations`.

  Download URL & SHA-256: jakarta-annotations-tck-3.0.0.zip, SHA-256: `8b916f1b4aed828337bd88b34bb39b133f04611c2dfe71541c2ec5d2dd22cd54`

- Jakarta Concurrency 3.1.1 TCK

  See test results in section `concurrency`.

  Download URL & SHA-256: concurrency-tck-3.1.1.zip, SHA-256: `22728d729f620d6a85ae903e7d1184e0a7508a4328491b785f1b4f3d7215ca93`

- Jakarta Contexts and Dependency Injection 4.1.0 TCK

  See test results in sections `cdi` and `cdi-lang-model`.

  Download URL & SHA-256: cdi-tck-4.1.0-dist.zip, SHA-256: `566c547e1a9c66792eefcc6feafea87ab0c0f2e3f71385bf96865359a685df00`

- Jakarta Dependency Injection 2.0.2 TCK

  See test results in section `di`.

  Download URL & SHA-256: jakarta.inject-tck-2.0.2-bin.zip, SHA-256: `23bce4317ca061c3de648566cdf65c74b57e1264d6891f366567955d6b834972`

- Jakarta Faces 4.1.1 TCK

  See test results in section `jsf`.

  Download URL & SHA-256: jakarta-faces-tck-4.1.1.zip, SHA-256: `e2024aabb0c784d23c076165a4f16e03892cc68c06f67517eec5ba66b9fb17f0`

- Jakarta JSON Binding 3.0.0 TCK

  See test results in section `jsonb`.

Download URL & SHA-256: jakarta-jsonb-tck-3.0.0.zip, SHA-256: `954fd9a3a67059ddeabe5f51462a6a3b542c94fc798094dd8c312a6a28ef2d0b`

- Jakarta JSON Processing 2.1.1 TCK

  See test results in section `jsonp`.

  Download URL & SHA-256: jakarta-jsonp-tck-2.1.1.zip, SHA-256: `949f203de84deffa8c7892b555918e42f1dd220ccb7b6800741ea58af62737c1`

- Jakarta RESTful Web Services 4.0.0 TCK

  See test results in section `jaxrs`.

  Download URL & SHA-256: jakarta-restful-ws-tck-4.0.0.zip, SHA-256: `9494caaf67a3e5524a5ee1d6e4b749782ca1cbc920c6451583b6249e51c4ac18`

- Jakarta Security 4.0.0 TCK

  See test results in section `security`.

  Download URL & SHA-256: jakarta-security-tck-4.0.0.zip, SHA-256: `e21a3d8251bf954c99a3f9a5f71911696509a1ed40c23673760785a3b88e4172`

- Jakarta WebSocket 2.2.0 TCK

  See test results in section `websocket`.

  Download URL & SHA-256: jakarta-websocket-tck-2.2.0.zip, SHA-256: `ff50825305bd1318a4e6bdfab21b829cfd8cf89fab8a970f007b52636d3f2629`

- Jakarta Pages 4.0.0 TCK

  See test results in section `pages`.

  Download URL & SHA-256: jakarta-pages-tck-4.0.0.zip, SHA-256: `5446aa866601a7c4c425f74054590cfc7f4fc05592a572d399ecc8694e265489`

- Jakarta Servlet 6.1.0 TCK

  See test results in section `servlet`.

  Download URL & SHA-256: jakarta-servlet-tck-6.1.0.zip, SHA-256: `1170697a87622a920bd50f0a68ed594c6efc5e17dd370566312194c4e12bec29`

- Jakarta Expression-Language 6.0.0 TCK

  See test results in section `expression-language`.

  Download URL & SHA-256: jakarta-expression-language-tck-6.0.0.zip, SHA-256: `90cc60b9d3340675e3224d9c4ab7a82b3cf63857f39f0006e143277eaa82bbce`

- Jakarta Tags 3.0.1 TCK

  See test results in section `tags`.

  Download URL & SHA-256: jakarta-tags-tck-3.0.1.zip, SHA-256: `b779d125ea36480b206629830ca80c1e506f65e6e8235b22e846a9cb40aaba04`

- Jakarta Validation 3.1.1 TCK

  See test results in section `validation`.

  Download URL & SHA-256: validation-tck-dist-3.1.1.zip, SHA-256: `bda9de47960845a7c73aa9a1616e4bbed3e346de3516f768819cf5af77bf969f`

- Java runtime used to run the implementation:

  `OpenJDK Runtime Environment zulu21.0.7 (build 21.0.7+6)`

- Summary of the information for the certification environment:

  Apache Derby, Linux, Ubuntu 20.04 LTS

Test Results

annotations

```
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

authentication

```
Completed running 163 tests.
Number of Tests Passed      = 163
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

cdi-langmodel

```
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

cdi-platform-tck

```
Completed running 1769 tests.
Number of Tests Passed      = 1769
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

cdi-tck

```
Completed running 1334 tests.
Number of Tests Passed      = 1334
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

concurrency

```
Completed running 197 tests.
Number of Tests Passed      = 183
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 14
```

data

```
Completed running 99 tests.
Number of Tests Passed      = 99
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

dsol-platform-tck

```
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

ejb30-platform-lite-tck

```
Completed running 1760 tests.
Number of Tests Passed      = 1760
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

ejb32-platform-tck-lite-timer-basic

```
Completed running 40 tests.
Number of Tests Passed      = 40
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

ejb

```
ejb32-lite-timer-interceptor

Completed running 28 tests.
Number of Tests Passed      = 28
Number of Tests Failed      = 0
Number of Tests with Errors = 0

ejb32-lite-timer-schedule:
Completed running 344 tests.
Number of Tests Passed      = 344
Number of Tests Failed      = 0
Number of Tests with Errors = 0

ejb32-lite-timer-service:

Completed running 8 tests.
Number of Tests Passed      = 8
Number of Tests Failed      = 0
Number of Tests with Errors = 0

ejb32-lite-timer-timerconfig:
Completed running 36 tests.
Number of Tests Passed      = 36
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

expression-language-platform-tck

```
Completed running 717 tests.
Number of Tests Passed      = 717
Number of Tests Failed      = 0
```

```
    Number of Tests with Errors = 0
```

expression-language-tck

```
    Completed running 361 tests.
    Number of Tests Passed      = 361
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

faces-tck

```
    Completed running 314 tests.
    Number of Tests Passed      = 297
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 17

    old-tck:
    Completed running 5389 tests.
    Number of Tests Passed      = 5389
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

inject-tck

```
    Completed running 50 tests.
    Number of Tests Passed      = 50
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

jdbc-platform-tck

```
    Completed running 2462 tests.
    Number of Tests Passed      = 2462
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

jsonb-platform-tck

```
    Completed running 10 tests.
    Number of Tests Passed      = 10
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

jsonb-tck

```
    Completed running 295 tests.
    Number of Tests Passed      = 290
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 5
```

jsonp

```
platform-tck:
Completed running 38 tests.
Number of Tests Passed      = 38
Number of Tests Failed      = 0
Number of Tests with Errors = 0


glassfish-runner:
Completed running 179 tests.
Number of Tests Passed      = 179
Number of Tests Failed      = 0
Number of Tests with Errors = 0


glassfish-runner-pluggability:
Completed running 18 tests.
Number of Tests Passed      = 18
Number of Tests Failed      = 0
Number of Tests with Errors = 0

payara-runner:
Completed running 196 tests.
Number of Tests Passed      = 196
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

pages-platform-tck

```
Completed running 18 tests.
Number of Tests Passed      = 18
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

pages-tck

```
Completed running 682 tests.
Number of Tests Passed      = 682
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

persistence-platform-tck

```
Completed running 3818 tests.
Number of Tests Passed      = 3818
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

rest

```
platform-tck:
Completed running 129 tests.
Number of Tests Passed      = 129
```

```
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    payara-se-tests:
    Completed running 7 tests.
    Number of Tests Passed      = 7
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    payara-server-remote-tests:
    Completed running 2667 tests.
    Number of Tests Passed      = 2607
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 60
```

security-tck

```
    Completed running 133 tests.
    Number of Tests Passed      = 133
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    old-tck
    Completed running 83 tests.
    Number of Tests Passed      = 83
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

servlet-tck

```
    Completed running 1716 tests.
    Number of Tests Passed      = 1716
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

signature-tck

```
    Completed running 2 tests.
    Number of Tests Passed      = 2
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

tags-tck

```
    Completed running 541 tests.
    Number of Tests Passed      = 541
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

transaction-platform-tck

```
    Completed running 154 tests.
```

```
Number of Tests Passed      = 154
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

websocket-platform-tck

```
Completed running 12 tests.
Number of Tests Passed      = 12
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

websocket-tck

```
Completed running 737 tests.
Number of Tests Passed      = 737
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

7.0.0

Platform TCK Results

As required by the Eclipse Foundation Technology Compatibility Kit License, following is a summary of the TCK results for Payara Server releases for Jakarta EE Platform 11.

Payara Server 7.0.0 Jakarta EE 11 Platform Certification Summary

As required by the Eclipse Foundation Technology Compatibility Kit License, following is a summary of the TCK results for Payara Server releases for Jakarta EE Platform 11.

- Product Name, Version and download URL (if applicable):

Payara Server 7.0.0

- Specification Name, Version and download URL:

  Jakarta EE Platform 11

  Jakarta EE Platform TCK 11.0.1.zip , SHA-256: `b2c0ad6db0514b75ff612dd7d855a3976c8ae9f240a3b690a684f61bf503be`

`ad`

- Public URL of TCK Results Summary:

  7.0.0 Platform TCK Results

  - Jakarta Activation 2.1.2

    See test results in section `activation`.

    Download URL & SHA-256: activation-tck-2.1.2.zip, SHA-256: `6c4aad27e45761dd9f3e0f8506f37edea41f42401465db750689145718b27a0b`

  - Jakarta Authentication 3.1.0

    See test results in section `authentication`.

    Download URL & SHA-256: jakarta-authentication-tck-3.1.0.zip, SHA-256: `8b916f1b4aed828337bd88b34bb39b133f04611c2dfe71541c2ec5d2dd22cd54`

  - Jakarta Annotations 3.0.0

    See test results in section `annotations`.

    Download URL & SHA-256: jakarta-annotations-tck-3.0.0.zip, SHA-256: `8b916f1b4aed828337bd88b34bb39b133f04611c2dfe71541c2ec5d2dd22cd54`

  - Jakarta Batch 2.1.5 TCK

    See test results in section `batch`.

    Download URL & SHA-256: jakarta.batch.official.tck-2.1.5.zip, SHA-256: `0dd8ca0f35cc696ea86d0dffaa1301cf2786806832ea1b2a491d528eaa57b3b7`

  - Jakarta Concurrency 3.1.1 TCK

    See test results in section `concurrency`.

    Download URL & SHA-256: concurrency-tck-3.1.1.zip, SHA-256:

`22728d729f620d6a85ae903e7d1184e0a7508a4328491b785f1b4f3d7215ca93`

- Jakarta Contexts and Dependency Injection 4.1.0 TCK

  See test results in sections `cdi` and `cdi-lang-model`.

  Download URL & SHA-256: cdi-tck-4.1.0-dist.zip, SHA-256: `566c547e1a9c66792eefcc6feafea87ab0c0f2e3f71385bf96865359a685df00`

- Jakarta Debugging Support for Other Languages 2.0.0 TCK

  See test results in section `debugging`.

  Download URL & SHA-256: jakarta-debugging-tck-2.0.0.zip, SHA-256: `71999815418799837dc6f3d0dc40c3dcc4144cd90c7cdfd06aa69270483d78bc`

- Jakarta Dependency Injection 2.0.2 TCK

  See test results in section `di`.

  Download URL & SHA-256: jakarta.inject-tck-2.0.2-bin.zip, SHA-256: `23bce4317ca061c3de648566cdf65c74b57e1264d6891f366567955d6b834972`

- Jakarta JSON Binding 3.0.0 TCK

  See test results in section `jsonb`.

  Download URL & SHA-256: jakarta-jsonb-tck-3.0.0.zip, SHA-256: `954fd9a3a67059ddeabe5f51462a6a3b542c94fc798094dd8c312a6a28ef2d0b`

- Jakarta JSON Processing 2.1.1 TCK

  See test results in section `jsonp`.

  Download URL & SHA-256: jakarta-jsonp-tck-2.1.1.zip, SHA-256: `949f203de84deffa8c7892b555918e42f1dd220ccb7b6800741ea58af62737c1`

- Jakarta Faces 4.1.1 TCK

  See test results in section `jsf`.

  Download URL & SHA-256: jakarta-faces-tck-4.1.1.zip, SHA-256: `153184ad38e522c45e74e5da7dd01b1f3be485f56e241a72728caa43479b07ec`

- Jakarta Mail 2.1.0 TCK

  See test results in section `javamail`.

  Download URL & SHA-256: jakarta-mail-tck-2.1.0.zip, SHA-256: `6f02a92e0a5ef60260e65f95938cc566da2f93a3d269c3b321da0d787a3448a5`

- Jakarta RESTful Web Services 4.0.0 TCK

  See test results in section `jaxrs`.

  Download URL & SHA-256: jakarta-restful-ws-tck-4.0.0.zip, SHA-256: `9494caaf67a3e5524a5ee1d6e4b749782ca1cbc920c6451583b6249e51c4ac18`

- Jakarta Security 4.0.0 TCK

  See test results in section `security`.

  Download URL & SHA-256: jakarta-security-tck-4.0.0.zip, SHA-256: `e21a3d8251bf954c99a3f9a5f71911696509a1ed40c23673760785a3b88e4172`

- Jakarta Authorization 3.0.0 TCK

  See test results in section `authorization`.

  Download URL & SHA-256: jakarta-authorization-tck-3.0.0.zip, SHA-256: `70f2e15334cfc525fa3b2ab575a96bd06cde942302d45d12da040242ff969fe8`

- Jakarta Tags 3.0.1 TCK

  See test results in section `tags`.

Download URL & SHA-256: jakarta-tags-tck-3.0.1.zip, SHA-256: `b779d125ea36480b206629830ca80c1e506f65e6e8235b22e846a9cb40aaba04`

- Jakarta Validation 3.1.1 TCK

  See test results in section `validation`.

  Download URL & SHA-256: validation-tck-dist-3.1.1.zip, SHA-256: `bda9de47960845a7c73aa9a1616e4bbed3e346de3516f768819cf5af77bf969f`

- Jakarta WebSocket 2.2.0 TCK

  See test results in section `websocket`.

  Download URL & SHA-256: jakarta-websocket-tck-2.2.0.zip, SHA-256: `ff50825305bd1318a4e6bdfab21b829cfd8cf89fab8a970f007b52636d3f2629`

- Jakarta Pages 4.0.0 TCK

  See test results in section `pages`.

  Download URL & SHA-256: jakarta-pages-tck-4.0.0.zip, SHA-256: `5446aa866601a7c4c425f74054590cfc7f4fc05592a572d399ecc8694e265489`

- Jakarta Servlet 6.1.0 TCK

  See test results in section `servlet`.

  Download URL & SHA-256: jakarta-servlet-tck-6.1.0.zip, SHA-256: `1170697a87622a920bd50f0a68ed594c6efc5e17dd370566312194c4e12bec29`

- Jakarta Persistence 3.2.0 TCK

  See test results in section `persistence`.

  Download URL & SHA-256: jakarta-persistence-tck-3.2.0.zip, SHA-256: `db74101c3f52f737f57804271ea9c845cb1941f2a0685284c54c4ad4fa1f05c7`

- Jakarta Expression-Language 6.0.0 TCK

  See test results in section `expression-language`.

  Download URL & SHA-256: jakarta-expression-language-tck-6.0.0.zip, SHA-256: `90cc60b9d3340675e3224d9c4ab7a82b3cf63857f39f0006e143277eaa82bbce`

- Jakarta Data 1.0.1 TCK

  See test results in section `data`.

  Download URL & SHA-256: data-tck-1.0.1.zip, SHA-256: `92babb604b4e346f8d19929979c1d6aaf013fdf990b5e97bdd95502ad7db5b4c`

- Java runtime used to run the implementation:

  `OpenJDK Runtime Environment zulu21.0.7 (build 21.0.7+6)`

- Summary of the information for the certification environment:

  Apache Derby, Linux, Ubuntu 20.04 LTS

Test Results

activation

```
Completed running 91 tests.
Number of Tests Passed      = 91
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests Not Run     = 0
Pluggability:
Completed running 2 tests.
Number of Tests Passed      = 2
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests Not Run     = 0
```

annotations

```
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

assembly

```
Completed running 12 tests.
Number of Tests Passed      = 12
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

authentication

```
Completed running 163 tests.
Number of Tests Passed      = 163
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

authorization

```
Completed running 53 tests.
Number of Tests Passed      = 53
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

batch

```
Completed running 390 tests.
Number of Tests Passed      = 372
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 18
```

cdi-lang-model

```
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

cdi-platform-tck

```
Completed running 1904 tests.
Number of Tests Passed      = 1904
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

cdi

```
Completed running 1334 tests.
Number of Tests Passed      = 1334
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

concurrency

```
Completed running 295 tests.
Number of Tests Passed      = 268
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 27
```

connector

```
Completed running 441 tests.
Number of Tests Passed      = 441
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

data

```
Completed running 99 tests.
Number of Tests Passed      = 99
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

dsol

```
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

ejb30-platform

```
assembly-tck:
Completed running 46 tests.
Number of Tests Passed      = 46
Number of Tests Failed      = 0
Number of Tests with Errors = 0

lite-tck:
Completed running 1760 tests.
Number of Tests Passed      = 1760
Number of Tests Failed      = 0
Number of Tests with Errors = 0

misc-tck:
Completed running 98 tests.
Number of Tests Passed      = 98
Number of Tests Failed      = 0
Number of Tests with Errors = 0

bb-tck:
Completed running 1081 tests.
Number of Tests Passed      = 1081
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

```
timer-tck:
Completed running 178 tests.
Number of Tests Passed      = 178
Number of Tests Failed      = 0
Number of Tests with Errors = 0

tx-tck:
Completed running 84 tests.
Number of Tests Passed      = 84
Number of Tests Failed      = 0
Number of Tests with Errors = 0

zombie-tck:
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

ejb32-platform-tck

```
lite-timer:
Completed running 2 tests.
Number of Tests Passed      = 2
Number of Tests Failed      = 0
Number of Tests with Errors = 0

lite-timer-basic:
Completed running 40 tests.
Number of Tests Passed      = 40
Number of Tests Failed      = 0
Number of Tests with Errors = 0

lite-timer-interceptor:
Completed running 60 tests.
Number of Tests Passed      = 60
Number of Tests Failed      = 0
Number of Tests with Errors = 0

lite-timer-schedule:
Completed running 536 tests.
Number of Tests Passed      = 536
Number of Tests Failed      = 0
Number of Tests with Errors = 0

lite-timer-service:
Completed running 8 tests.
Number of Tests Passed      = 8
Number of Tests Failed      = 0
Number of Tests with Errors = 0

lite-timer-timerconfig:
Completed running 36 tests.
Number of Tests Passed      = 36
Number of Tests Failed      = 0
Number of Tests with Errors = 0

mdb:
Completed running 1 tests.
```

```
    Number of Tests Passed      = 1
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    relaxedclientview:
    Completed running 21 tests.
    Number of Tests Passed      = 21
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

expression-language-platform

```
    Completed running 717 tests.
    Number of Tests Passed      = 717
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

expression-language-tck

```
    Completed running 361 tests.
    Number of Tests Passed      = 361
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

faces-tck

```
    Completed running 314 tests.
    Number of Tests Passed      = 314
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 17

    old-tck:
    Completed running 5389 tests
    Number of tests failed 0
    Number of tests with errors 0
    Number of tests skipped 0
```

inject-tck

```
    Completed running 50 tests.
    Number of Tests Passed      = 50
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

integration-tck

```
    Completed running 15 tests.
    Number of Tests Passed      = 15
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

javaee-module-tck

```
Completed running 24 tests.
Number of Tests Passed      = 24
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

jdbc-platform-tck

```
Completed running 4924 tests.
Number of Tests Passed      = 4924
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

jsonb-platform

```
Completed running 18 tests.
Number of Tests Passed      = 18
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

jsonb-tck

```
Completed running 295 tests.
Number of Tests Passed      = 290
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 5
```

jsonp-platform-tck

```
Completed running 76 tests.
Number of Tests Passed      = 76
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

jsonp-tck

```
glassfish-runner:
Completed running 179 tests.
Number of Tests Passed      = 179
Number of Tests Failed      = 0
Number of Tests with Errors = 0

glassfish-runner-pluggability:
Completed running 18 tests.
Number of Tests Passed      = 18
Number of Tests Failed      = 0
Number of Tests with Errors = 0

payara-runner:
Completed running 196 tests.
Number of Tests Passed      = 196
```

```
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

javamail

```
    Completed running 112 tests.
    Number of Tests Passed      = 112
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

messaging-platform-tck

```
    Completed running 3510 tests.
    Number of Tests Passed      = 3489
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 21
```

mvc-tck

```
    Completed running 135 tests.
    Number of Tests Passed      = 135
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

pages-platform-tck

```
    Completed running 28 tests.
    Number of Tests Passed      = 28
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

pages-tck

```
    Completed running 682 tests.
    Number of Tests Passed      = 682
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

persistence

```
    core1:
    Completed running 238 tests.
    Number of Tests Passed      = 238
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    core2:
    Completed running 112 tests.
    Number of Tests Passed      = 112
    Number of Tests Failed      = 0
```

```
    Number of Tests with Errors = 0

    core3:
    Completed running 521 tests.
    Number of Tests Passed      = 520
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 1

    core4:
    Completed running 405 tests.
    Number of Tests Passed      = 404
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 1
```

persistence-platform-tck

```
    annotations:
    Completed running 984 tests.
    Number of Tests Passed      = 984
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    basic:
    Completed running 12 tests.
    Number of Tests Passed      = 12
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    callback:
    Completed running 396 tests.
    Number of Tests Passed      = 396
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    core-entitytest:
    Completed running 904 tests.
    Number of Tests Passed      = 894
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 10

    criteria-api:
    Completed running 3726 tests.
    Number of Tests Passed      = 3726
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    derived-id:
    Completed running 72 tests.
    Number of Tests Passed      = 72
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    ee:
    Completed running 180 tests.
    Number of Tests Passed      = 162
    Number of Tests Failed      = 0
```

```
    Number of Tests with Errors = 0
    Number of Tests skipped     = 18

    entity-graph:
    Completed running 78 tests.
    Number of Tests Passed      = 78
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    entity-manager-2:
    Completed running 132 tests.
    Number of Tests Passed      = 132
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    entity-manager:
    Completed running 200 tests.
    Number of Tests Passed      = 200
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    entity-manager-factory:
    Completed running 42 tests.
    Number of Tests Passed      = 42
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    entity-manager-factory-close-exceptions:
    Completed running 6 tests.
    Number of Tests Passed      = 6
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    entity-transaction:
    Completed running 20 tests.
    Number of Tests Passed      = 20
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    entitytest:
    Completed running 96 tests.
    Number of Tests Passed      = 96
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    enums:
    Completed running 317 tests.
    Number of Tests Passed      = 317
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    exceptions:
    Completed running 34 tests.
    Number of Tests Passed      = 34
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    inheritance:
    Completed running 60 tests.
    Number of Tests Passed      = 60
```

```
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    jpa:
    Completed running 4 tests.
    Number of Tests Passed      = 4
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    jpa22:
    Completed running 94 tests.
    Number of Tests Passed      = 94
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    lock:
    Completed running 36 tests.
    Number of Tests Passed      = 36
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    metamodel-api:
    Completed running 1560 tests.
    Number of Tests Passed      = 1560
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    nested-embedding:
    Completed running 18 tests.
    Number of Tests Passed      = 18
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    override:
    Completed running 156 tests.
    Number of Tests Passed      = 156
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    query:
    Completed running 1242 tests.
    Number of Tests Passed      = 1242
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    relationship:
    Completed running 198 tests.
    Number of Tests Passed      = 198
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    stored-procedure-query:
    Completed running 238 tests.
    Number of Tests Passed      = 238
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    types:
    Completed running 306 tests.
    Number of Tests Passed      = 306
```

```
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    unit-util:
    Completed running 18 tests.
    Number of Tests Passed      = 18
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    util:
    Completed running 6 tests.
    Number of Tests Passed      = 6
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    versioning:
    Completed running 6 tests.
    Number of Tests Passed      = 6
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    various-tck:
    Completed running 685 tests.
    Number of Tests Passed      = 685
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

rest

```
    platform
    Completed running 129 tests.
    Number of Tests Passed      = 129
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    payara-se-tests
    Completed running 7 tests.
    Number of Tests Passed      = 7
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    payara-server-remote-tests
    Completed running 2667 tests.
    Number of Tests Passed      = 2607
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 60
```

security

```
    Completed running 133 tests.
    Number of Tests Passed      = 133
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    Old TCK:
    Completed running 83 tests.
```

```
Number of Tests Passed      = 83
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

servlet

```
Completed running 1716 tests.
Number of Tests Passed      = 1716
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

signature

```
Completed running 4 tests.
Number of Tests Passed      = 4
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

soap

```
Completed running 447 tests.
Number of Tests Passed      = 447
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

tags

```
Completed running 541 tests.
Number of Tests Passed      = 541
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

transaction-platform-tck

```
Completed running 195 tests.
Number of Tests Passed      = 195
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

websocket-platform-tck

```
Completed running 12 tests.
Number of Tests Passed      = 12
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

websocket-tck

```
Completed running 737 tests.
Number of Tests Passed      = 737
```

```
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

xa-platform-tck

```
    Completed running 66 tests.
    Number of Tests Passed      = 66
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

xml-binding-tck

```
    Completed running 24624 tests.
    Number of Tests Passed      = 24624
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

xml-ws-tck

```
    Completed running 954 tests.
    Number of Tests Passed      = 954
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

Web TCK Results

As required by the Eclipse Foundation Technology Compatibility Kit License, following is a summary of the TCK results for Payara Server releases for Jakarta EE Platform 11.

Payara Server 7.0.0 (Web Profile) Jakarta EE 11 Web Profile Certification Summary

- Product Name, Version and download URL (if applicable):

Payara Server 7.0.0 (Web Profile)

- Specification Name, Version and download URL:

  Jakarta EE Web Profile 11

- TCK Version, digital SHA-256 fingerprint and download URL:

Jakarta EE Platform TCK 11.0.1.zip , SHA-256: `b2c0ad6db0514b75ff612dd7d855a3976c8ae9f240a3b690a684f61bf503bead`

- Public URL of TCK Results Summary:

  7.0.0 Web TCK Results

- Any Additional Specification Certification Requirements:
  - Jakarta Authentication 3.1.0

    See test results in section `authentication`.

    Download URL & SHA-256: jakarta-authentication-tck-3.1.0.zip, SHA-256: `9421c6ca66274d32dfb408848f75a42d57f120599fe0d8403c5c5c1141d5ac4d`

  - Jakarta Annotations 3.0.0

    See test results in section `annotations`.

    Download URL & SHA-256: jakarta-annotations-tck-3.0.0.zip, SHA-256: `8b916f1b4aed828337bd88b34bb39b133f04611c2dfe71541c2ec5d2dd22cd54`

  - Jakarta Concurrency 3.1.1 TCK

    See test results in section `concurrency`.

    Download URL & SHA-256: concurrency-tck-3.1.1.zip, SHA-256: `22728d729f620d6a85ae903e7d1184e0a7508a4328491b785f1b4f3d7215ca93`

  - Jakarta Contexts and Dependency Injection 4.1.0 TCK

    See test results in sections `cdi` and `cdi-lang-model`.

    Download URL & SHA-256: cdi-tck-4.1.0-dist.zip, SHA-256: `566c547e1a9c66792eefcc6feafea87ab0c0f2e3f71385bf96865359a685df00`

- Jakarta Dependency Injection 2.0.2 TCK

  See test results in section `di`.

  Download URL & SHA-256: jakarta.inject-tck-2.0.2-bin.zip, SHA-256: `23bce4317ca061c3de648566cdf65c74b57e1264d6891f366567955d6b834972`

- Jakarta Faces 4.1.1 TCK

  See test results in section `jsf`.

  Download URL & SHA-256: jakarta-faces-tck-4.1.1.zip, SHA-256: `e2024aabb0c784d23c076165a4f16e03892cc68c06f67517eec5ba66b9fb17f0`

- Jakarta JSON Binding 3.0.0 TCK

  See test results in section `jsonb`.

  Download URL & SHA-256: jakarta-jsonb-tck-3.0.0.zip, SHA-256: `954fd9a3a67059ddeabe5f51462a6a3b542c94fc798094dd8c312a6a28ef2d0b`

- Jakarta JSON Processing 2.1.1 TCK

  See test results in section `jsonp`.

  Download URL & SHA-256: jakarta-jsonp-tck-2.1.1.zip, SHA-256: `949f203de84deffa8c7892b555918e42f1dd220ccb7b6800741ea58af62737c1`

- Jakarta RESTful Web Services 4.0.0 TCK

  See test results in section `jaxrs`.

  Download URL & SHA-256: jakarta-restful-ws-tck-4.0.0.zip, SHA-256: `9494caaf67a3e5524a5ee1d6e4b749782ca1cbc920c6451583b6249e51c4ac18`

- Jakarta Security 4.0.0 TCK

  See test results in section `security`.

Download URL & SHA-256: jakarta-security-tck-4.0.0.zip, SHA-256: `e21a3d8251bf954c99a3f9a5f71911696509a1ed40c23673760785a3b88e4172`

- Jakarta WebSocket 2.2.0 TCK

  See test results in section `websocket`.

  Download URL & SHA-256: jakarta-websocket-tck-2.2.0.zip, SHA-256: `ff50825305bd1318a4e6bdfab21b829cfd8cf89fab8a970f007b52636d3f2629`

- Jakarta Pages 4.0.0 TCK

  See test results in section `pages`.

  Download URL & SHA-256: jakarta-pages-tck-4.0.0.zip, SHA-256: `5446aa866601a7c4c425f74054590cfc7f4fc05592a572d399ecc8694e265489`

- Jakarta Servlet 6.1.0 TCK

  See test results in section `servlet`.

  Download URL & SHA-256: jakarta-servlet-tck-6.1.0.zip, SHA-256: `1170697a87622a920bd50f0a68ed594c6efc5e17dd370566312194c4e12bec29`

- Jakarta Expression-Language 6.0.0 TCK

  See test results in section `expression-language`.

  Download URL & SHA-256: jakarta-expression-language-tck-6.0.0.zip, SHA-256: `90cc60b9d3340675e3224d9c4ab7a82b3cf63857f39f0006e143277eaa82bbce`

- Jakarta Tags 3.0.1 TCK

  See test results in section `tags`.

  Download URL & SHA-256: jakarta-tags-tck-3.0.1.zip, SHA-256: `b779d125ea36480b206629830ca80c1e506f65e6e8235b22e846a9cb40aaba04`

- Jakarta Validation 3.1.1 TCK

  See test results in section `validation`.

  Download URL & SHA-256: validation-tck-dist-3.1.1.zip, SHA-256: `bda9de47960845a7c73aa9a1616e4bbed3e346de3516f768819cf5af77bf969f`

- Java runtime used to run the implementation:

  `OpenJDK Runtime Environment zulu21.0.7 (build 21.0.7+6)`

- Summary of the information for the certification environment:

  Apache Derby, Linux, Ubuntu 20.04 LTS

Test Results

annotations

```
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

authentication

```
Completed running 163 tests.
Number of Tests Passed      = 163
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

cdi-langmodel

```
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

cdi-platform-tck

```
Completed running 1769 tests.
Number of Tests Passed      = 1769
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

cdi-tck

```
Completed running 1334 tests.
Number of Tests Passed      = 1334
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

concurrency

```
Completed running 197 tests.
Number of Tests Passed      = 183
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 14
```

data

```
Completed running 99 tests.
Number of Tests Passed      = 99
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

dsol-platform-tck

```
Completed running 1 tests.
Number of Tests Passed      = 1
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

ejb30-platform-lite-tck

```
Completed running 1760 tests.
Number of Tests Passed      = 1760
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

ejb32-platform-tck-lite-timer-basic

```
Completed running 40 tests.
Number of Tests Passed      = 40
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

ejb

```
ejb32-lite-timer-interceptor

Completed running 28 tests.
Number of Tests Passed      = 28
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

```
    ejb32-lite-timer-schedule:
    Completed running 344 tests.
    Number of Tests Passed      = 344
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    ejb32-lite-timer-service:

    Completed running 8 tests.
    Number of Tests Passed      = 8
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    ejb32-lite-timer-timerconfig:
    Completed running 36 tests.
    Number of Tests Passed      = 36
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

expression-language-platform-tck

```
    Completed running 717 tests.
    Number of Tests Passed      = 717
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

expression-language-tck

```
    Completed running 361 tests.
    Number of Tests Passed      = 361
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

faces-tck

```
    Completed running 314 tests.
    Number of Tests Passed      = 297
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 17

    old-tck:
    Completed running 5389 tests.
    Number of Tests Passed      = 5389
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

inject-tck

```
    Completed running 50 tests.
    Number of Tests Passed      = 50
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

jdbc-platform-tck

```
Completed running 2462 tests.
Number of Tests Passed      = 2462
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

jsonb-platform-tck

```
Completed running 10 tests.
Number of Tests Passed      = 10
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

jsonb-tck

```
Completed running 295 tests.
Number of Tests Passed      = 290
Number of Tests Failed      = 0
Number of Tests with Errors = 0
Number of Tests skipped     = 5
```

jsonp

```
platform-tck:
Completed running 38 tests.
Number of Tests Passed      = 38
Number of Tests Failed      = 0
Number of Tests with Errors = 0


glassfish-runner:
Completed running 179 tests.
Number of Tests Passed      = 179
Number of Tests Failed      = 0
Number of Tests with Errors = 0


glassfish-runner-pluggability:
Completed running 18 tests.
Number of Tests Passed      = 18
Number of Tests Failed      = 0
Number of Tests with Errors = 0

payara-runner:
Completed running 196 tests.
Number of Tests Passed      = 196
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

pages-platform-tck

```
Completed running 18 tests.
Number of Tests Passed      = 18
```

```
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

pages-tck

```
    Completed running 682 tests.
    Number of Tests Passed      = 682
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

persistence-platform-tck

```
    Completed running 3818 tests.
    Number of Tests Passed      = 3818
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

rest

```
    platform-tck:
    Completed running 129 tests.
    Number of Tests Passed      = 129
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    payara-se-tests:
    Completed running 7 tests.
    Number of Tests Passed      = 7
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    payara-server-remote-tests:
    Completed running 2667 tests.
    Number of Tests Passed      = 2607
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
    Number of Tests skipped     = 60
```

security-tck

```
    Completed running 133 tests.
    Number of Tests Passed      = 133
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0

    old-tck
    Completed running 83 tests.
    Number of Tests Passed      = 83
    Number of Tests Failed      = 0
    Number of Tests with Errors = 0
```

servlet-tck

```
Completed running 1716 tests.
Number of Tests Passed      = 1716
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

signature-tck

```
Completed running 2 tests.
Number of Tests Passed      = 2
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

tags-tck

```
Completed running 541 tests.
Number of Tests Passed      = 541
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

transaction-platform-tck

```
Completed running 154 tests.
Number of Tests Passed      = 154
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

websocket-platform-tck

```
Completed running 12 tests.
Number of Tests Passed      = 12
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

websocket-tck

```
Completed running 737 tests.
Number of Tests Passed      = 737
Number of Tests Failed      = 0
Number of Tests with Errors = 0
```

Eclipse MicroProfile Certification

Eclipse Microprofile Certification Summary

Payara Server is a certified Eclipse Microprofile implementation. As proof of compatibility, output of MicroProfile Compatibility Test Suite tests are presented for

each certified version in each release.

Security

Overview

This section describes Payara's official policy for managing Azul Payara's security vulnerabilities.

The CVE Program

Payara, a leading provider of Jakarta EE and MicroProfile runtimes, is authorized by the Common Vulnerabilities and Exposures (CVE) Program as a CVE Numbering Authority (CNA).

Payara will publish authoritative cybersecurity vulnerability information about its products via the CVE Program. Vulnerabilities will be given a unique, alphanumeric identifier, building the CVE List that feeds into the U.S. National Vulnerability Database (NVD), and playing a role in the CVE Program's mission to identify, define and catalogue cybersecurity vulnerabilities.

NOTE	The CVE Program is sponsored by the Cybersecurity and Infrastructure Security Agency (CISA) of the U.S. Department of Homeland Security (DHS).

Developers using Azul Payara products will benefit from the collaboration, as vulnerabilities will be part of the standardized and publicly disclosed CVE List. This will result in time and cost savings for those using Payara products, as security issues can be discussed, dealt with and prevented through the use of a trusted, standardized catalogue.

In general, Payara follows the CVE Program's CNA Operational rules to make decisions on reported vulnerabilities and CVE record management.

General Guidelines

- Payara is committed to fixing any vulnerabilities and exploits identified in any Azul Payara products and distributions.
- Any reported vulnerabilities will be fixed with celerity as soon as they are reported and confirmed.
- Security vulnerabilities will have to be verified to affect an Azul Payara product before getting a CVE ID assigned.
- Vulnerabilities reported on product versions that are no longer supported due to having reached their End-of-Support will be disclosed but will NOT be fixed.
- Vulnerabilities not related to Azul Payara will be ignored and not disclosed as CVE records. These include:
  - Operating system vulnerabilities
  - JVM-specific vulnerabilities
  - Conditions or behaviors that do not lead to a security impact.
    - These include an increase in access for an attacker, a decrease in availability of a target, or a violation of an implemented security policy.
  - Physical attacks unless there is a specific security feature, supported claim, or policy that defends against the physical attack.
  - Well-documented or commonly understood non-default configuration or runtime changes made by an authorized user
  - Brute-force denial-of-service (DoS) and brute-force resource exhaustion attacks, for example, high-rate or high-bandwidth denial-of-service attacks.
    - Failure to implement common defenses against such attacks will also be determined NOT to be valid vulnerabilities.
- Payara Enterprise customers can request security fixes to be back-ported to any supported releases (if allowed by their corresponding Enterprise support lifecycle) used on their environments.

Vulnerabilities on Third-Party Components

As Azul Payara depends on third-party components of other software packages like Apache Guava, Jackson, Smack, etc. Vulnerabilities on specific versions of these components may be reported as well. Payara is committed to analysing any reported vulnerabilities and upgrading them to a patched version whether applicable.

In these cases, reports SHOULD if possible reference an existing CVE record that details the vulnerability in question.

Reporting Security Vulnerabilities

To report a vulnerability, make an official request through the Customer Support Portal. Please make sure to describe the vulnerability encountered in great detail, along with detailed steps on how to reproduce or exploit the vulnerability and environmental details like:

- Azul Payara distribution (i.e, Server, Micro, Embedded, Docker Image) and version
- Operating System
- JDK Version

IMPORTANT	If the report is related to a Vulnerabilities on Third-Party Components, please include the CVE ID of the vulnerability that affects the third-party component if you have access to it.

CVE ID Assignment

Once a report it's received, we'll determine if it exposes a valid vulnerability that needs to be disclosed and assign a CVE ID to it as established by the CNA Program rules. We'll assign a CVE ID when:

1. Payara reasonable evidence to determine the existence of a vulnerability on the report.
2. The vulnerability has been or is expected to be **publicly disclosed**.
3. The vulnerability has the potential to cause significant harm.

If these conditions are not met, Payara will not address the vulnerability and will NOT assign a CVE ID.

NOTE	If a report includes evidence of multiple **independently fixable** vulnerabilities, Payara will assign separate CVE IDs for each confirmed vulnerability in the report.

CVE Record Disclosure

Once a security vulnerability is fixed and published in an official release of Azul Payara, the following will occur:

- The details of the vulnerability will be published in the release notes in this documentation site.
- The CVE record will be officially published.
- Customers will be notified of any necessary mitigation measures to be considered if they are affected by the vulnerability.

CVE Record Rejection

In some cases, the investigation of a vulnerability that has a CVE ID assigned may yield a result where the vulnerability in question does not warrant a fix (it cannot be reproduced, it doesn't affect Azul Payara, is not considered as harmful as originally reported).

In cases like this, the CVE record will be rejected along with a reason explaining why the vulnerability didn't require disclosure.

CVE Disputes

If as a reporter, you do not agree with the assessment that Payara has made on a reported vulnerability when a CVE has not been assigned or an assigned CVE has been rejected, you may let us know through the corresponding customer portal ticket. We'll follow the CVE Program Policy and Procedure for Disputing a CVE Record to handle the resolution of this dispute within our scope as a CNA.

Payara will be open to negotiating any disputes and reconsider our position on previous assessments if enough evidence and information is provided by the reported.

Security Advisories

All security reports, that correspond to either fixes to vulnerabilities that affect Azul Payara directly or vulnerabilities to its third-party dependencies are catalogued under the "Security Advisories" page.

Each vulnerability entry will contain:

- Its CVE ID
- The CVE record's CVSS Score
- The status of the vulnerability (`FIXED`, `REJECTED`, `N/A`)
- The summary of the exploit
- The release where a fix was published (if applicable)
- Additional observations on the resolution of the vulnerability.

Credits

If you are interested in being credited as the finder and/or reporter of the vulnerability, please let us know explicitly on your report and provide the following details:

- Your **name** or an **alias**.
- *(Optional)* The name of an organisation you belong to and wants to be credited.

See Also

- CVE Website
- Payara's CNA Partner Profile
- CNA Operational Rules
- CVE Program Glossary

Security Advisories

The following is a list of reported ***C**ommon **V**ulnerabilities and **E**xposures* that have been analyzed and documented across releases:

Non-CVE Vulnerabilities

Here is a collection of historic non-**CVE** vulnerabilities that may warrant attention:

Reference	Status	Summary	Release	Observations
FISH-13363	FIXED	Restrict Access To Vulnerable EL Expressions	7.0.0	Fixed by upgrading jsftemplating
Payara Enterprise Support Ticket	FIXED	Vulnerability in Metro's WSDL Code Importing/Parsing - Remote Code Execution	5.28.0	Recommended to immediately upgrade to this release if using any JAX-WS features in applications deployed in public-facing environments.
OWASP Docs	FIXED	Web administration console is vulnerable against clickjacking/UI redress attacks.	4.1.2.174	
Payara Support Ticket	FIXED	Under some circumstances authenticated caller/user identities get confused.	4.1.1.171.11	

Reference	Status	Summary	Release	Observations
Payara Support Ticket	FIXED	CORBA security context gets corrupted under certain conditions	4.1.2.181.2, 4.1.2.182, 5.182	

Appendix

Schemas

Payara Schemas

Schemas for Azul Payara.

- "DTD for payara-web.xml^"
- "DTD for payara-resources.xml^"
- "DTD for payara-ejb-jar.xml^"
- "DTD for payara-application.xml^"
- "DTD for payara-application-client.xml^"
- "DTD for payara-acc.xml^"