# Things to Consider When Migrating from JBoss EAP to Payara® Server Enterprise 5

Migration Guide

The Payara® Platform - Production-Ready,
Cloud Native and Aggressively Compatible.

# Contents

# Introduction

Migrating applications from JBoss **E**nterprise **A**pplication **P**latform (EAP) to Payara Server 5 can be a simple and straightforward process because both servers rely on the Jakarta EE (Java EE) specifications. However, there are differences in many areas because many Java EE APIs in JBoss EAP and Payara Server Enterprise are implemented by different components. Moreover, the configuration of certain aspects like external resources, high-availability and deployment is not covered by any specification and is, in fact, very different in both servers. This guide gives you an overview of the differences between JBoss EAP and Payara Server, which components and features in Payara Server are equivalent to those in JBoss EAP, and how to configure some frequently-used resources and features in Payara Server compared to how you're used to them in JBoss EAP.

Payara Server is originally based on GlassFish Server and they both still share a lot of code, concepts, and ecosystem tools. Since Sun Microsystems first developed GlassFish as the reference implementation (RI) of Java EE, it has been used by both developers in its open source form, and by Sun - then, later, by Oracle - in production, fully supported. In a competitive market, Oracle GlassFish fared well and became a popular choice, with the vast majority using the GlassFish Server Open Source Edition. In late 2013, Oracle announced the end of commercial support for GlassFish.

One year after the original announcement of the end of support from Oracle, Steve Millidge - a Java EE expert and the founder of Payara Services Ltd - announced the arrival of Payara Server and, with it, a reintroduction of a fully supported option with a very similar platform and the added reassurance of regular releases, bug fixes, patches, and enhancements.

Payara Server evolved rapidly from just a "supported GlassFish alternative" to a fully Open Source product that can run Java EE and Jakarta EE applications. It contains several unique features in the areas of monitoring and security, for example, and was the first run-time that integrates Java EE and Eclipse MicroProfile®. Starting in June 2020, Payara Server has split into two editions:

- **Payara Server Community**, which is geared towards innovation and rapid development, is completely free to use in development environments. The purpose of this edition is to allow the general developer community to rapidly adopt new features that might not be suitable for production environments until they are stable enough. This edition is directly comparable to JBoss WildFly, as both products fill the same niche.
- **Payara Server Enterprise**, which is tailored for use in production environments for extended periods of time with guaranteed stability. A Payara Enterprise subscription is needed to run the server in production environments and customers with an active subscription get access to premium support and specific feature sets that are not available in the Community Edition. Innovative features are adopted at a much slower pace in this edition, since they will only be integrated when they are stable enough. This edition is directly comparable to JBoss EAP, as they are premium products with commercial support. Commercially licensed Payara Server Enterprise focuses on the requirements and needs of the customers to optimize their deployment and management scenarios, and the system stability at run-time.

To simplify things, the rest of this document will strictly reference Payara Server Enterprise edition, since it is the recommended product to run in a production environment.

## Releases

JBoss EAP Server development started in 1999 under the name of EJB-OSS. It was then renamed a few times, to JBOSS, JBoss, later to JBoss AS (JBoss Application Server). Years later (2014), a new fork of the project was created in the name of WildFly to clearly distinguish itself from JBoss EAP. Compared to JBoss EAP, WildFly aims to evolve faster and is suitable for fast development of applications with expected shorter maintenance periods. On the other side, JBoss EAP adopts new features more slowly, mostly after they are proven within WildFly, and is more suitable for projects that require long maintenance periods and long-term stability.

JBoss EAP releases are tied to long development software cycles with major versions of the product coming out aligned with big structural changes (like a new version of the Java EE/Jakarta EE specification). Minor updates are released and aligned with newly introduced features and are usually synchronized with a corresponding WildFly community release as well. Patches that contain security and vulnerability fixes are released frequently and on-demand, depending on the customer base needs. This is important, since JBoss EAP main's concern is to establish a robust middleware platform that can be used reliably on production environments.

In a similar vein, Payara Server Enterprise will adopt releases of Jakarta EE and MicroProfile APIs when they have been implemented in a stable manner in the Community edition and can be ported with minimal risks. At the time of this writing, the latest release of Payara Server Enterprise, 5.20.0 supports both Jakarta EE 8 and MicroProfile 3.3.

Payara Server Enterprise supports reliable and secure deployments of Jakarta EE (Java EE) applications in any environment: on premise, in the cloud, or hybrid. Automatic security alerts and fixes along with monthly releases, bug fixes, and patches ensure the stability of your environment and a 10-year software lifecycle means you don't have to worry about upgrading a year or two after implementing the software. Unlike other companies providing support services through an outsourced helpdesk, Payara Server Enterprise customers receive support directly from the engineers.

# JBoss EAP vs. Payara Server

Although Payara Server Enterprise offers similar features as JBoss EAP, they are often based on different technologies and concepts, and often use different terminology. Therefore, before going into migration from JBoss EAP to Payara Server, we'll provide you with an overview of similar features and concepts that exist in both products. The following table compares concepts and mechanisms in JBoss EAP and how they equate to those in Payara Server:

| JBoss EAP | Payara Server Enterprise | Description |
|---|---|---|
| Standalone mode | Domain with a single server | Environment with a single server serving requests, no clustering |
| Managed domain | Domain with multiple instances | Environment with multiple server instances in a cluster |
| Server Group | Deployment Group | A virtual group of servers under a managed domain |
| Infinispan | Domain Data Grid (Hazelcast) | Technology used for data replication and caching |
| Domain Controller | Domain Admin Server | The admin server managing server instances in a domain |
| Host Controller | Node (no controller) | A helper service (in JBoss EAP) or a configuration setting (in Payara Server) used to manage remote server instances |
| Web Console | Admin Console | Web interface to access administration server |
| JBoss CLI | Asadmin CLI | Command line tool to run administration commands |
| HTTP Management Interface | REST Management Interface | HTTP (REST) interface to access administration server |
| Server | Standalone Instance | A separate server instance that is managed by the administration server |
| Alternative Configuration Files | Named Configuration | Different configuration for server instances |
| Security Domain, Realm | Realm, Login Module | Server-side authentication and authorization components |

Under the hood, Payara Server Enterprise contains a lot of different technologies that provide the same Java/Jakarta EE APIs. In terms of compatibility with Java/Jakarta EE versions, use the following table to compare which product versions are compatible with each other:

| Java / Jakarta EE Version | Payara Server Compatible Versions | JBoss EAP Compatible Versions |
|---|---|---|
| Java EE 7 | Payara Server Enterprise 4.1.21.151 - 4.1.2.191 | JBoss EAP 7.0 |
| Java EE 8 | Payara Server 5.181 - 5.192<br>Payara Server Enterprise 5.20.0 | JBoss EAP 7.2 |
| Jakarta EE 8 | Payara Server 5.193.1 - 5.201 (Last public release before Enterprise)<br>Payara Server Enterprise 5.20.0 | JBoss EAP 7.3 |

While JBoss EAP is mainly composed of multiple Red Hat projects, Payara Server is solely based on components that are Java EE reference implementations and now most of them are under the auspices of the Eclipse Foundation. However, there are shared components between the two products as they are both Java EE reference implementations as well. For the purposes of modern application development, we'll refer to these APIs in their "current" Jakarta EE state as both products fully support them.

Here's the list of the most often used Jakarta EE APIs and respective technologies in JBoss EAP and Payara Server:

| Jakarta EE API | Payara Server Enterprise | JBoss EAP |
|---|---|---|
| CDI | Weld | Weld |
| JSF | Mojarra | Mojarra |
| Bean Validation | Hibernate Validator | Hibernate Validator |
| JSON Binding | Yasson | Yasson |
| Jakarta Persistence (JPA) | EclipseLink | EclipseLink |
| WebSocket | Tyrus | Undertow |
| RESTful Web Services (JAX-RS) | Jersey | RESTEasy |
| Servlet | Grizzly | Undertow |

| | | |
|---|---|---|
| Batch | Jbatch | Jberet |
| Jakarta Messaging (JMS) | Open MQ | Artemis |
| Jakarta Security | Soteria | Elytron |
| XML Web Services (JAX-WS) | Metro | Apache CXF |

In addition to Jakarta EE APIs, both JBoss EAP and Payara Server Enterprise also provide MicroProfile APIs. While JBoss EAP relies on API implementations from the SmallRye's RedHat project, Payara Server Enterprise provides its own implementation for all MicroProfile APIs supported by its current release.

## Administration

Both JBoss EAP and Payara Server Enterprise offer multiple management interfaces:

- Web Console
- Command Line Interface (CLI)
- HTTP Management Interface

The Web Console in Payara Server Enterprise is called the Admin Console and is accessible using a standard admin HTTP port, by default at http://localhost:4848. By default, it doesn't require any authentication and is only accessible to local clients (not over the network). Remote access is allowed only when SSL encryption is enabled and a non-empty password is set for the default admin user.

Command Line Interface for Payara Server Enterprise is called `Asadmin CLI`. It's a script utility located in the **bin** directory (**asadmin.sh** for unix-based systems, **asadmin.bat** for Windows) in a Payara Server Enterprise installation. This utility can manage a running instance of Payara Server and also supports commands to manage the lifecycle of existing domains and/or instances. For example, to start the Payara Server Enterprise default domain as a background service, run the **start-domain** command:

```
bin/asadmin start-domain
```

The HTTP Management Interface in Payara Server is mostly referred to as REST Management Interface. It's available at the same URL root as the Admin Console, at the **/management/domain** path. By default, the URL is http://localhost:4848/management/domain. Payara Server Enterprise contains a simple web client for the REST Management Interface, which is available at that URL by default. It allows browsing all the available management resources and commands in a browser. All management URLs also support JSON and XML content formats for programmatic access. The

output format is requested either by adding the format name to the path (e.g. http://localhost:4848/management/domain.json) or specifying it using the HTTP Accept request header.

Moreover, the **Asadmin Recorder** is a unique feature of Payara Server Enterprise which can help you write scripts for setting up your environment. It records the steps you perform within the Admin Console and saves them as their equivalent `AsAdmin` commands to a text file. This file can then be used to reproduce the configuration of your server in an automated manner to facilitate the deployment of your server in multiple environments.

## Operating Modes

JBoss EAP supports two operating modes:

- *Standalone*: Single server instance, operates independently, each standalone instance is configured separately.
- *Managed domain*: Allows running and managing a multi-server topology from a single **Domain Controller** server.

Payara Server Enterprise doesn't distinguish between these two modes. It always runs in **Domain mode**. A single **D**omain **A**dmin **S**erver (DAS) can act either a standalone server like JBoss EAP in standalone mode or as a central domain controller with additional managed server instances.

Payara Server Enterprise can run as a standalone server instance if you run only the DAS server without any other instances in the domain. A standalone DAS server offers a management interface and also supports application deployment. In this mode, all management commands automatically work on the DAS server, so there's no need to specify a command target. Applications are automatically deployed to the DAS server and any resources and configuration changes are targeted to the DAS server by default. Initially, each domain is configured to have a single DAS without any additional instances. Therefore to run Payara Server Enterprise as a standalone server, it's enough to just start the domain from the command line:

```
bin/asadmin start-domain
```

You can create and manage multiple domains that have a single DAS and run them independently as standalone servers. Such independent DAS servers can still be configured to join the same **Data Grid** to allow the exchange of data if needed.

A default Payara Server domain that has a single DAS can be extended to have one or more standalone server instances. These domain instances are managed from the DAS admin server which acts as a **central domain controller**. Domain Instances can run on the same physical machine or on a different machine and communicate with the DAS over the network. Domain instances automatically join the same data grid and form an HA cluster without any additional configuration. Domain instances that run on the same host are grouped into a node and can communicate with the DAS

directly unlike JBoss EAP managed instances which communicate with the Domain Controller via a separate **host controller** process.

Domain instances can each have their own configuration or share a common configuration. In both cases, the configuration is managed in the DAS which distributes configuration changes to running instances or instances update it from the DAS at startup.

Domain instances can form a deployment group to manage them together or use them as a command target. For example, an application can be deployed to all instances in a deployment group with a single command, or all instances in the same deployment group can be launched or stopped with a single command. It's recommended that all instances in the same deployment group share the same configuration to have access to the same resources. This configuration can be parameterized for each instance to account for any differences like port bindings.

A single Payara Server Enterprise installation can contain configuration for multiple domains. It's possible to run multiple domains at once if they use a different set of ports. Such domains are independent of each other, they only share the same installation files. However, more frequently, multiple domains are useful to have different domain configurations and switch between them when needed. This can be useful for example to switch between configurations during development or run different test suites in different configurations. As an example, the default Payara Server Enterprise installation comes with two domains available out of the box. The default domain called `domain1` is tuned for development while the `production` domain is tuned for running applications in production.

## Clustering and High Availability

The basic concepts of clustering in Payara Server Enterprise are very similar to those in JBoss EAP. The following clustering components are equivalent:

| JBoss EAP | Payara Server Enterprise | Description |
|---|---|---|
| Domain Controller | DAS | Main administration server and coordinator |
| Managed Server | Standalone Instance | A server instance managed by the administration server |
| Cache Container | Domain Data Grid | Distributed and replicated memory cluster |
| Server Group | Deployment Group | A logical group of managed servers used as deployment targets |

Payara Server Enterprise's Domain Data Grid is a similar concept to JBoss EAP's *Infinispan* subsystem. It provides a backbone for distributed memory stores and caches for instances in the Payara Server domain. It's flexible and can scale up and down easily just by starting or stopping additional server instances on the network. While JBoss EAP allows the configuration of multiple cache containers for a single domain arrangement but have to be prepared before being used, Payara Server Enterprise provides a single Data Grid configured out of the box and ready to be distributed to all instances in an existing domain.

*Deployment Groups* in Payara Server Enterprise are a similar concept to *Server Groups* used by JBoss EAP's Managed domain configuration and offer similar functionality. All instances in a deployment group can be managed together, e.g. started and stopped at the same time and applications can be deployed or undeployed on all of them in a single step.

## Security

Payara Server Enterprise provides multiple mechanisms to secure access to its server runtime and applications running on it on multiple levels:

- SSL/TLS network encryption using generated self-signed and standard CA-trusted certificates.
- Authentication using several built-in mechanisms, with the capability to create customer authentication mechanisms.
- Mapping of user groups to application-specific roles, as mandated by the JACC specification.
- Securing applications based on user roles.
- Auditing service that records administration activity for auditing purposes.

Payara Server Enterprise provides a self-signed certificate out of the box for every domain that is created and it can also generate new self-signed certificates on demand. This self-signed certificate is used by default for all encrypted communication, e.g. for the default HTTPS listener and the secured RMI/IIOP channel. Payara Server Enterprise also ships with multiple public certificates for known trusted certification authorities (CA), to make the validation of chain certificates a bit easier.

Authentication and authorization in Payara Server Enterprise is provided by security realms. They are used to authenticate incoming requests using an associated JAAS login module. After a login module authenticates a request, it often retrieves information about the user's roles from the realm that delegate its control to it.

In contrast, JBoss EAP uses security realms to provide authentication and authorization *mainly for securing administration management interfaces*. As a compliment, it introduces the concept of **security domains** that are used to associate multiple login modules and realms with deployed applications. In JBoss EAP, login modules are not called by realms but the other way around. A security domain associated with an application delegates to one or more login modules for authentication. At least one of the login modules in a security domain is usually a `RealmDirect` module, which then

delegates to a specific realm. On the other hand, Payara Server Enterprise only allows associating a single realm directly with an application so the realm takes the role of a security domain. However, thanks to the capabilities of the Jakarta EE Security API, Payara Server Enterprise can be extended so that multiple realms can be associated with an application since the server runtime exposes each security realm as a Jakarta EE standard **Identity Store**.

The management interfaces of both JBoss EAP and Payara Server are secured by a file-based mechanism. JBoss EAP uses the `ManagementRealm` security realm which is file-based and Payara Server Enterprise uses the `admin-realm` realm which is an instance of a `FileRealm` definition (identities are stored in a encoded file in local storage).

The following table summarizes how basic security concepts in JBoss EAP map to those in Payara Server Enterprise in summary form:

| JBoss EAP | Payara Server Enterprise |
|---|---|
| Security Domain | Realm |
| Login Module | Realm |
| Security Realm | `FileRealm` |
| `ManagementRealm` realm | `admin-realm` realm |

In addition to this, Payara Server Enterprise has introduced multiple utilities that can be used from the command line to manage the lifecycle of certificate entries in a domain keystore and/or truststore. These utilities are not present in the Community Edition and are considered a premium feature.

# Clustering in Payara Server Enterprise vs. JBoss EAP

Clustering in JBoss EAP is composed of:

- A **Domain Controller** server managing several server instances in a managed domain configuration
  - These instances talk to the controller via their respective **Host Controller**.
- The **Infinispan** subsystem that manages and coordinates distributed memory stores available to other runtime services.

Clustering in Payara Server Enterprise works in a similar (albeit with crucial differences) manner:

- A **DAS** (Domain Administration Server) managing several server instances that belong to the same domain
  - These instances talk to the DAS directly, without the assistance of a separate process.
- The **Domain Data Grid** (based on Hazelcast) that manages and coordinates distributed an in-memory store and data coordinator that is available to other runtime services
  - Bonus: The Data Grid transcends domain-registered instances. Other domains can join the data grid with the right discovery mode settings and can exchange data with each other if needed.

Subsystems providing distributed memory and caching are also similar in concept. Both are based on embedded in-memory data grid solutions that are well integrated.

Infinispan within JBoss EAP is an in-memory data grid and distributed caching solution. Its main purpose is:

- Caching: Store data in a distributed (using different nodes) in-memory cache so that retrieval is faster than from remote sources. This is a typical optimization for slowly changing data sets.
- Transactions: Infinispan can be used in a transactional way.
- Events: Events can be distributed between different nodes triggering listener code.

Infinispan can also be used as JCache (JSR-107) implementation too, allowing applications to access these data stores using a standard API. It is important to note that JCache is not yet an official part of the Jakarta EE specification, so it is not a truly portable and standard solution.

The same functionality is available within Payara Server Enterprise with the help of the **Domain Data Grid** (powered by Hazelcast Open Source edition):

- Data can be cached using a manual operation (basically, change the statements *CacheFactory.getCache* to *hazelcastInstance.getMap* for example) or use the JCache option backed by the Hazelcast library.
- Hazelcast has support for the Jakarta Transaction API (JTA) specification. This means you can put data into the grid in a transactional way. It can even join in a distributed XA transaction as a party.
- Just as in the case of Infinispan, events can be fired which are triggering listener code on remote nodes. Payara Server Enterprise takes advantage of this feature to allow that applications trigger standard CDI events as remote events that can be observed by other standalone instances in the data grid
  - This is a proprietary feature of the Payara Platform's Public API which allows sending and receiving CDI events outside of the same JVM context.

Both Infinispan and Payara Server's Domain Data Grid provide querying, distributed processing, off-heap storage and support cloud environments of many major cloud providers. Some of the more advanced features like off-heap storage or WAN replication is only available to Payara Server Enterprise customers with the purchase of a separate license from Hazelcast Enterprise edition.

Payara Server Enterprise is fully compatible with this edition, so both products can be used in conjunction for powerful data-replication measures.

This all means that all use cases for which you are using Infinispan functionality can be mapped in a one to one way to the corresponding Hazelcast functionality which is included by default with Payara Server Enterprise.

On the differences: Payara Server Enterprise standalone instances don't use separate host controller equivalents to coordinate with the DAS. Instead, the instances either just start and connect to the DAS or the DAS manages them remotely over SSH or by using the Docker API directly, without any intermediary service. The DAS is able to install and start instances this way on remote hosts without any user intervention if it has the necessary permissions over the host machines. The DAS uses a concept called Node which is a mere configuration grouping-component to store information for accessing and managing a remote host machine and all standalone instances that reside on it.

## Overview of Clustering in Payara Server Enterprise

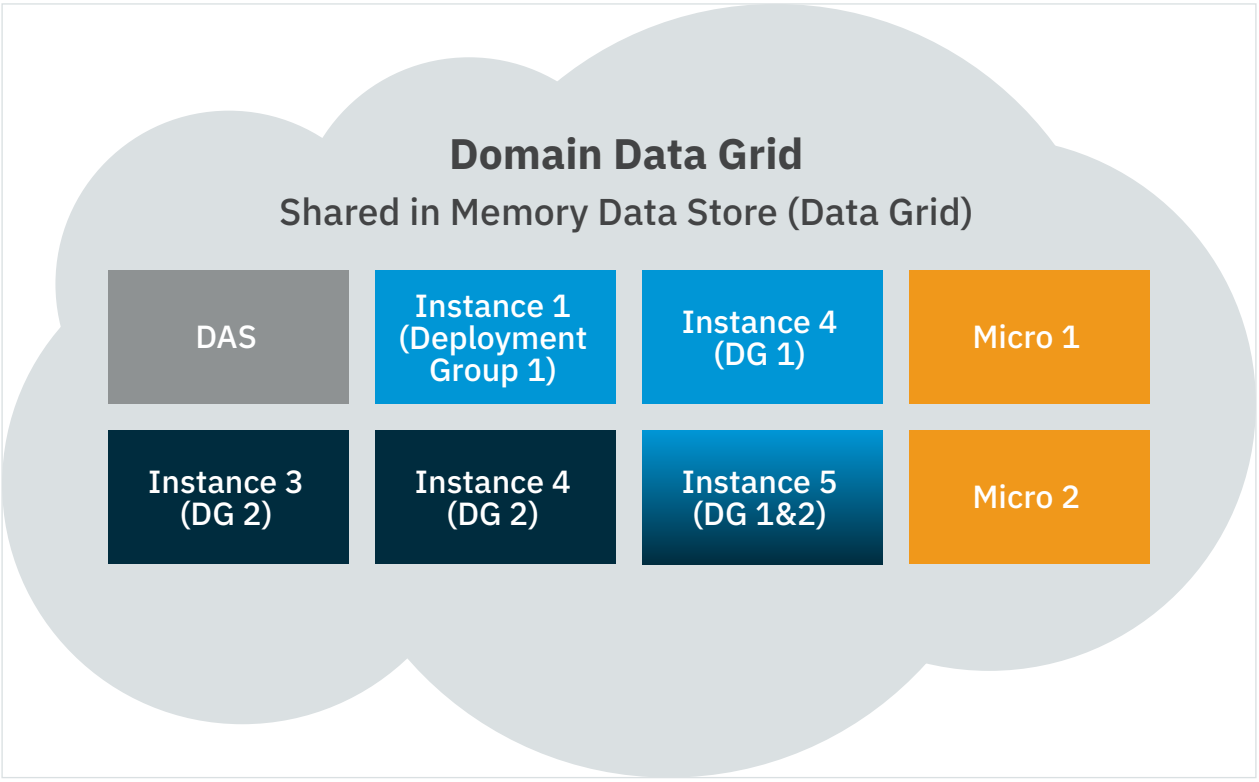
Payara Server Enterprise supports creating cluster instances out of the box, without any additional configuration or support tools. To create a new cluster member, just create an additional standalone instance using the Domain Admin Server (DAS). The DAS and all running instances automatically join the Data Grid upon being started.

Clustering in Payara Server Enterprise has two levels. On the first level, all running instances join the same data grid, which means they participate in sharing and replicating distributed memory data. As the data grid is powered by Hazelcast, it relies on its specific discovery mechanisms that allow its members to detect other members that want to join the grid and allow or reject its operation. The server runtime relies on a custom discovery mechanism called `domain`, which instructs instances that want to join the Data Grid to search for the DAS and coordinate with it its inclusion in the grid. Other mechanisms are supported by default:

- `multicast`, which relies on the multicast protocol to allow multiple instances to join together. This discovery mechanism treats all instances equally and hence makes no distinction between DAS and stand-alone instances. As such, when interested in conforming a cluster of multiple Payara Server Enterprise domains, this is the way to go. Due to the nature of this mechanism,  it is the default discovery mode used by Payara Micro Enterprise.
- `tcp-ip`, which relies on all instances knowing the IP addresses of all members of the grid in advance to allow them to talk to each other.
- `dns`, which functions in a similar manner as the previous mechanism but relies on DNS names instead.

If needed, Payara Server Enterprise can also be extended by using customized Hazelcast discovery modes for more complex scenarios as well. To do this, you'll have to provide a custom Hazelcast configuration files and set it up in the DAS.

The Domain Data Grid is flexible and can scale up and down easily just by starting or stopping additional instances on the network. Moreover, a single Domain Data Grid can be joined not only by Payara Server instances but also by Payara Micro instances. Payara Micro Enterprise is a separate runtime distribution of the Payara Platform designed to run microservice oriented architectures simply and efficiently. Payara Micro can thus be used to run several smaller services together with the main application in the same grid, effectively sharing memory and communicating together using the same clustering infrastructure.




On the second level, chosen domain instances can form a deployment group. All instances in a deployment group can be managed together, e.g. started and stopped together and applications and resources can be deployed or undeployed on them in a single-step operation.

Domain instances can be run locally or on remote hosts. Instances running on the same host are grouped under a node. There are four types of nodes according to how the communication to their host machines is managed by the DAS:

- SSH - instances are started by the DAS over an SSH connection. Payara Server Enterprise installation files are copied over the network if required.
- DCOM (deprecated) - similar to the previous mode but instead relies on the DCOM protocol, which is available on Windows systems only.
- CONFIG - instances are not contacted by the DAS. They are started separately and contact the DAS via its management port.
- DOCKER - instances are created in a Docker container that will live in the host machine and the DAS coordinates its lifecycle using the Docker Engine REST Admin interface.

## High Availability Support

Payara Server Enterprise supports multiple failover mechanisms to ensure that applications will be highly available:

- HTTP Session Replication - HTTP Session data is available on all instances in the Domain Data Grid.
- Stateful EJB Replication - Stateful EJB data is available on all instances in the Domain Data Grid.
- Message Queue Broker connection failover - connection failover to another broker in the cluster.
- Single Sign-on state failover - SSO data is available on all instances in the Domain Data Grid.
- RMI-IIOP failover - a remote EJB call fails over to another instance in a deployment group.
- Distributed application-level cache - a caching API to store and retrieve data from a cache distributed and replicated across multiple instances in the same cluster (Data Grid).
- JPA Second-level cache - distributed cache of JPA entities synchronized by either using a mechanism on top of the distributed application-level cache or via JMS brokers.

Payara Server also supports RMI-IIOP load balancing to distribute IIOP client requests to remote EJBs evenly across a deployment group. HTTP requests can be load balanced using an external proxy server like Apache HTTP or Nginx, with either session replication or sticky sessions. Payara Server Enterprise doesn't support load balancing guided by **server-side load balance factors**, which are supported in JBoss EAP through the `mod_cluster` subsystem.

Most of the replication and data synchronization across the Payara Server instances uses the Data Grid, which allows server instances flexibly to form a grid of interconnected "nodes" that exchange data with each other. Each instance offers some memory to store the distributed information and replicas of data stored in other instances. This makes the data grid robust and resilient to data losses as data is recreated from replicas whenever an instance is disconnected and its data lost.

# Migrating Configuration of Server Resources

Most often, the first thing you need to migrate when porting your application from JBoss EAP to Payara Server Enterprise is the configuration of resources provided by the server and consumed by the application. Even if resources like JDBC datasources, mail sessions and security realms are often used by the applications in a standard and portable way, they often need to be configured in the server's runtime outside of the applications. These configuration settings are different between JBoss EAP and Payara Server Enterprise and can't be just copied during migration. The following sections describe what to expect when migrating the configuration of these type of resources to Payara Server Enterprise.

## Datasources

Datasources are by far the most frequently used resources defined in JBoss EAP outside of applications. To create a data source you have to:

- Provide a JDBC (4.0) driver to the server's runtime.
- Define a datasource that references the driver and configure its connection pool.
- Associate a JNDI name to the datasource to make it available to other components and applications.

The recommended way to install a JDBC driver in JBoss EAP is to deploy it as a regular JAR artifact. When you run JBoss EAP in domain mode, the JAR file will be distributed to all servers in the domain as with other regular deployments.

In Payara Server Enterprise, JDBC connection pools for datasources are configured separately. A datasource in Payara Server is just a JNDI resource definition that exposes a referenced JDBC connection pool to applications as a DataSource object under a specified JNDI name. In Payara Server Enterprise, the following is needed to configure a datasource:

- Provide a JDBC (4.0) driver to the server's runtime.
- Define a JDBC connection pool that references the driver and configure its access to the corresponding database.
- Define a JDBC resource with a custom JNDI name that references the JDBC connection pool.

In Payara Server Enterprise, the recommended way to install the JDBC driver is to add its JAR file as a domain-shared library. If you have multiple instances in a Payara Server Enterprise domain, this JAR file will be distributed to them automatically in the same way so the instances have access to the classes in this library. In general, the amount of work done in Payara Server to install and distribute a JDBC driver JAR is the same. But the way how it works is different. In Payara Server, unlike in JBoss EAP, the JAR won't appear in the list of deployed modules. It will be simply added to the classpath for all applications to access it. This is enough for the connection pool to access and use the driver.

To work with custom JAR files in Payara Server Enterprise, the following Asadmin commands can be used:

- `add-library` - Used to “install” a library in a target domain
- `remove-library` - Used to “uninstall” a registered library in a target domain

For example, add the MySQL driver JAR to the default domain from the command line, run the following command:

```
asadmin add-library /path/to/mysql-driver.jar
```

After you install the JDBC driver, you can proceed to create a JDBC connection pool that uses the driver to connect to an existing MySQL database, and finally create a JDBC resource that exposes the pool as a datasource via JNDI.

## Security Realms

Jakarta EE applications are often secured by a mechanism specified by the realm name in the application deployment descriptors (like the standard `web.xml` configuration file). However, the application only specifies the name of the realm to use and the remaining configuration steps needs to be done in the server. This is due to the nature of how the JACC and JAAS specification delegate the definition of login modules and authentication mechanisms to the server runtime.

In the case of JBoss EAP, the realm name that is configured in a deployment descriptor is bound to a **security domain** that is responsible for authenticating the users. The security domain then delegates to multiple login modules and subsequently to user realms defined in the server. The security chain during authentication is as follows:

- The application is associated with a single security domain.
- The security domain delegates to one or more login modules.
- Each login module authenticates the user in order of definition.
- If the authentication succeeds, a user principal is retrieved with an optional list of user groups/roles.
    - If defined, RealmDirect login modules delegates to a realm to defer the authentication process.

Security realms in Payara Server Enterprise function in a different manner than JBoss EAP realms: They behave similarly to JBoss login modules. Payara Server Enterprise binds the application’s realm name directly to a single realm that is predefined the server. This, on one side, simplifies the configuration. On the other side, it’s much less flexible because only a single realm can be used with an application. The chosen realm delegates authentication and authorization to a JAAS login module bound by its name (in JAAS context). The security chain in Payara Server Enterprise is:

- The application is associated with a single security realm.
- The security realm delegates to a JAAS login module for authentication.
- If the authentication succeeds, a user principal is retrieved with an optional list of user groups/roles.

If a more complex authentication and authorization mechanism is required, a custom realm needs to be developed and installed into the server's domain. Alternatively, multiple realms can be associated with an application using the standard Jakarta EE Security API. Payara Server Enterprise exposes each security realm as an Identity Store that can be enabled in an application using specific annotation mechanisms provided by the Payara Platform's Public API.

If a security domain in JBoss EAP only uses a single login module that matches a realm in Payara Server Enterprise then migration is rather straightforward. This login module can be directly converted to a matching realm which should be given the same name as the security domain so that it matches the deployed application. However, there may be differences in how some matching realms are configured and behave. The following table compares realms in Payara Server to their login module equivalents in JBoss EAP:

| JBoss EAP login module(s) | Payara Server Enterprise security realm | Notes |
|---|---|---|
| `Certificate`, `CertificateUsers` | `CertificateRealm` | |
| `CertificateRoles` | `CertificateRealm` | CertificateRoles reads roles/group data from a file.<br><br>Payara Server Enterprise relies on role-to-group mapping rules defined in proprietary deployment descriptors (payara-web.xml). |
| `Database`, `DatabaseUsers` | `JDBCRealm` | JBoss EAP relies on a user-defined SQL query to read user and role data from the corresponding table.<br><br>Payara Server Enterprise on the other hand expects a pre-defined table structure, so table names and columns should be specified. |
| `Ldap`, `LdapUsers` | `LDAPRealm` | |
| `Simple`, `PropertiesUsers`, `UsersRoles`, `RealmDirect` | `FileRealm` | `FileRealm` stores user in a proprietary format in plain text (passwords are encoded) |

## JavaMail Sessions

Configuring a JavaMail session is rather straightforward in Payara Server Enterprise. JavaMail sessions are domain-level resources so they can be shared to every instance in the domain. To configure a session, you just need to create a **JavaMail session** resource and configure the following settings:

- Connection details and authentication credentials of the user that will connect to the SMTP server.
- JNDI name of the mail session.
- Default sender mail address.

This all can be done using the **create-javamail-resource** `Asadmin` command.

In the case of JBoss EAP, the configuration of a JavaMail session is split in two separate components:

- An outbound socket binding definition used to establish the SMTP connection.
- The actual JavaMail session resource associated with a JNDI name and bound to the previous socket definition.

# Migrating KeyCloak Configuration

Applications deployed on JBoss EAP are often secured using KeyCloak, which is a security framework that supports centralized Identity and Access Management and Single Sign-On features. Payara Server Enterprise allow deployed applications to integrate with separate instance of KeyCloak, as this is one of the advantages of the product. However, KeyCloak itself isn't supported in Payara Server Enterprise and can't be deployed to an existing domain, so if you have active installations of Keycloak on JBoss EAP there are several alternatives for migrating your security infrastructure over to Payara Server Enterprise:

## Configure Payara Server Enterprise to Use Your Existing KeyCloak Installation

The simplest solution is to continue using the same KeyCloak server running on JBoss EAP and configure your Payara Server Enterprise applications to integrate with it by using the KeyCloak servlet filter. This filter is provided by the framework and needs to be added to your web application as a standard servlet filter for all resources you need to secure. More information about this component can be found in the official KeyCloak documentation: https://www.keycloak.org/docs/latest/securing_apps/index.html#_servlet_filter_adapter.

## Migrate your Existing KeyCloak Installation to Run in Standalone Mode

Running Keycloak in standalone mode means that you can forego an installation of JBoss EAP and instead use the default installer that is built on top of WildFly. This simplifies the process and gives you complete access to your current KeyCloak installation, so consider to do this if you can't keep maintaining an existing installation that runs on JBoss EAP.

## Forego KeyCloak and Use the Payara Server Enterprise's Single Sign-On (SSO) Feature

Payara Server Enterprise provides a native Single Sign-On (SSO) feature. This feature can easily replace KeyCloak if all applications are deployed in the same Payara Server Enterprise domain and are bound to the same security realm. When SSO is enabled, a user that is authenticated by one application is then automatically authenticated in all other applications that are authenticated against the same security realm until they decide to log out (or their existing user sessions expire). This works by storing the security context after successful authentication and distributing it to all other applications as if the user logged in simultaneously to all the applications. Any authentication mechanism supported by Payara Server Enterprise, e.g. LDAP, can be used to authenticate users via the respective security realms.

## Replace KeyCloak with Another Identity and Access Management Solution

Another way to integrate existing applications with KeyCloak is to refactor your application's code-base to use OAuth2 authentication instead of the standard JAAS mechanisms and use KeyCloak as an OAuth2 provider. OAuth2 is supported in Payara Server Enterprise using the embedded OAuth2 authentication definition provided by the standard Jakarta Security API. You can also authenticate on the web page using an OAuth2 Javascript library and pass the acquired JWT token to the back-end services using the MicroProfile JWT authentication mechanism supported by Payara Server Enterprise. More about this specific feature can be found here: https://docs.payara.fish/documentation/payara-server/public-api/oauth-support.html and about MicroProfile JWT here: https://docs.payara.fish/documentation/microprofile/jwt.html

Payara Server Enterprise also supports integration with other Identity Management solutions similar to KeyCloak. As an example, you could replace from KeyCloak to Forge Rock OpenAM and follow this guide to set up OpenAM: https://blog.payara.fish/forgerock-integration-with-payara-server-part-1-installation

# Cloud Support

Containerized environments are becoming more important these days. JBoss EAP has its own official Docker images (for each release) which are available to customers that possess an active subscription.

Payara Server is no exception in the cloud evolution. We have also official Docker images that are available to customers in a private Docker registry. Payara Micro Enterprise, a specialized packaging of Payara Server which is optimized for cloud and micro-service environments is another product of the Payara Platform that you can easily use to create powerful microservice oriented architectures that are backed up by orchestration tools like Kubernetes.

Due to this correspondence, you can run Payara Server Enterprise on cloud providers like Azure, Amazon and Google, but also on RedHat OpenShift just as you can do with JBoss EAP.

# IDE Support

All major IDEs like IntelliJ® IDEA, Apache NetBeans, Eclipse® IDE, and Visual Studio Code have support for both servers. So you can use your favorite IDE to develop your application. Apache NetBeans supports Payara Server Enterprise out of the box and development of its integrations is supported by our developers as well. The plugins for the Eclipse IDE and Visual Studio Code are maintained by the Payara team. In IntelliJ IDEA, the plugin for GlassFish works equally well with Payara Server Enterprise, which a proper plugin coming in the horizon.

IDE plugins for Payara Server Enterprise support automatic redeployment when code is changed and compiled. This shortens the turnaround between a code change and the application being updated and ready to be tested.

Payara Server Enterprise also supports automatic deployment and redeployment of an application by dropping the application artifact in the autodeployment folder. This might be useful if no IDE can be used for automatic redeployment.

# Innovation

The team behind Payara Server is also actively working on the future of Java Enterprise applications: Payara Services is an Eclipse Foundation Solutions Member and a Strategic Member of the Jakarta EE working group, shaping the future of Eclipse Jakarta EE™ along with future versions of the platform.

We are also helping related technologies and frameworks. For example, we are also actively involved in the Eclipse MicroProfile specifications. We are not only defining the specifications together with the other involved parties, but Payara Server was one of the first servers which combined the MicroProfile implementations and the ability to run Java EE applications in one run-time. This gives you the ultimate flexibility to choose the right mix of dependencies for your use case and also allows you to implement a strategy of gradual migration from the Java EE platform to a more micro-service alike application structure.

For these microservice applications, Payara Server has a special packaging called Payara Micro. It provides Jakarta EE Web Profile specifications, some additional Jakarta EE specifications like Jakarta Concurrency, and MicroProfile specifications in a single jar file. This Hollow Jar deployment model (where the server is in one single jar and runs your application packaged as a war file) has many advantages over the Fat Jar approach. Each single application code change no longer results in the replacement of the layer in a Docker environment which contains also the server run-time. This reduces Image sizes, bandwidth usages and deployment times.

But enhancements are not only implemented outside the Java EE area. For example, Payara Server has many additions in the area of application security. Using the Security API (added in Java EE 8) there are various authentication and authorization schemes implemented like OAuth2, OpenIdConnect, and JWT tokens.

# Why Payara Server?

Payara Server is notably better then JBoss in the following areas:

## Production-Ready and Stable With Full Support

When you download the Payara Platform, you're downloading and adopting a production-ready version of Payara Server or Payara Micro. You can download a production-ready version from the download section of the website and you can find the complete source code on GitHub. With frequent community releases and a 10-year software lifecycle, a monthly release schedule for Payara Enterprise customers including bug fixes and patches, the Payara Platform offers security and stability without the need for upgrading every year or two. Payara Enterprise customers enjoy fast issue resolution directly from our global Engineers for both production and development issues along with priority on new features requests.

## Cloud-Native and Aggressively Compatible

Payara Server is optimized for production environments in any environment: cloud, on-premises, or hybrid. The Payara Platform is container-friendly, including Docker and Kubernetes, and compatible with services you're already using such as Microsoft Azure™, Amazon AWS, and MicroProfile. Our subscription costs are cloud-user friendly, and stay the same whether you're running your environment on-premise or on a public cloud.

## Open Source Software with a Future You Help Define

The Payara Platform is derived from GlassFish Open Source the Java EE reference implementation. Payara Enterprise customers are invited to customer advisory calls every six months to help drive the evolution of the Payara Platform, as new features and enhancements are developed to meet the needs of customers. Being open source, the Payara User Community allows you to submit your ideas, feedback, and collaboration to ensure Payara Server is the best option for production Jakarta EE applications and Payara Micro is the best option for running Jakarta EE applications in a modern virtualized infrastructure.



**sales@payara.fish**

**+44 207 754 0481**

**www.payara.fish**