Migrating from WildFly to Payara Community

What to Consider For a Successful Migration from WildFly to Payara® Platform Community

Power Up Your Jakarta EE

User Guide

Contents

Guide Updated: **December 2024**

Introduction

Migrating applications from WildFly to Payara Platform Community can be a seamless and straight-forward process, thanks to their shared compatibility with Jakarta EE (and Java EE) specifications. There are certainly distinctions between the two application server technologies. For example, many Java EE APIs are implemented by different components in WildFly and Payara Platform Community. Moreover, the configuration of certain aspects, such as external resources, high-availability and deployment, is not covered by any specification and is, in fact, very different in these servers. Nonetheless, the transition from WildFly to Payara Platform Community is not only possible but can also be exceptionally easy.

This guide aims to simplify the migration process, offering a clear comparison of WildFly's and Payara Platform Community's features and components, identifying equivalent tools and helping you bridge your existing WildFly expertise with Payara Platform Community's concepts. In addition, this document provides step-by-step guidance on how to configure frequently used resources and features in Payara Server.

About Payara Services and Payara Platform Community

Payara Services Ltd is a leading provider of application server technologies developed to address Jakarta EE application and user needs. Its technologies support monolith, microservices and hybrid solutions on premise, in the cloud as well in hybrid environments.

Payara Platform Community, which includes Payara Server Community and Payara Micro Community, is a streamlined and developer-focused, open-source and cloud-native application server technology that is designed for anyone experimenting with new features and APIs. Thanks to frequent releases, it regularly offers new features. As such, it is ideal for developers and software engineering students who want to innovate rapidly, test new functionalities and experiment.

For mission-critical applications in production environments as well as contexts that require robustness, security and regulatory compliance, Payara Services offers Payara Platform Enterprise, which has more operation-focused capabilities within its Payara Server Enterprise and Payara Micro Enterprise. The platform builds on the Community edition and adds capabilities suited for running mission-critical applications in production environments.

WildFly vs. Payara Platform Community

Releases

WildFly Server development started in 1999 under the name of EJB-OSS. It was then renamed a few times, to JBOSS, JBoss and later to JBoss AS (JBoss Application Server). It got its current name of WildFly in 2014 to clearly distinguish this solution from JBoss EAP (Enterprise Application Server), which is an application server based on WildFly and commercially supported by Red Hat.

Compared to JBoss EAP, WildFly aims to evolve faster and is suitable for rapid application developments that are expected to require shorter maintenance lifecycles. Conversely, JBoss EAP typically adopt new features more slowly, mostly after they are proven within WildFly. Also, it is more suitable for projects that require long maintenance lifecycles and long-term stability.

A new version of WildFly is released four times a year. Plus, there may be occasional patches, up to around five releases a year altogether. In comparison, Payara Platform Community follow a regular release cadence, with up to 12 releases per year.

Thanks to a such rapid release cycle, Payara Platform Community is fast to implement the latest version of Jakarta EE and MicroProfile APIs, which are usually featured in the next release after the APIs are released. The latest version of Payara Server 6 supports Jakarta EE 10 and MicroProfile 6.1. The following table showcases how quickly Payara Server adopted new versions of Jakarta EE and MicroProfile:

API version	API released	Payara Server Community Version	Payara Server Community Released
MicroProfile 6.0	December 2022	6.2023.3	March 2023
Jakarta EE 10	September 2022	6.2022.1	January 2022
MicroProfile 4.1	April 2021	5.2021.6	June 2021
MicroProfile 3.2	November 2019	5.194	December 2019
Jakarta EE 8	September 2019	5.193.1	October 2019
MicroProfile 2.1	October 2018	5.191	March 2019

Payara Platform Community supports deployments of Jakarta EE (Java EE) applications in any environment: on premise, in the cloud, or hybrid. When in need of high security and reliability for mission-critical applications in production environments, we recommend the use of fully supported Payara Platform Enterprise. Through this option, you can benefit from automatic security alerts and fixes along with regular releases, bug fixes and patches. These ensure the stability of your environment. These elements, combined with a 10-year software lifecycle, lead to peace of mind, as you don't have to worry about upgrading a year or two after implementing the software. Even more, anyone using the commercially supported edition, Payara Platform Enterprise, can benefit from direct access to technical support from Payara Services engineers.

Technology Comparison

While Payara Platform Community and WildFly offer similar features, they are typically built on different technologies and concept. In addition, they often use distinct terminology. To support your migration from WildFly to Payara Platform Community, we've outlined the comparable features and concepts between the two.

The following table compares the main concepts of WildFly and how they relate to concepts you'll find in Payara Platform Community:

Concept	WildFly	Payara Platform Community	Description
Operating Mode			
Single Server	Standalone mode	Domain with a single server	Environment with one server, no clustering
Multiple Servers	Managed domain	Domain with multiple instances	Environment with clustered server instances
Clustering			
Server Group	Server Group	Deployment Group	A logical grouping of servers within a managed domain
Data Replication/ Caching	Infinispan	Domain Data Grid (Hazelcast)	Technology used to synchronize data and improve performance

Concept	WildFly	Payara Platform Community	Description
Management			
Central Admin Server	Domain Controller	Domain Admin Server (DAS)	The server that manages all other instances in the domain
Remote Server Management	Host Controller	Node (no controller)	A service (WildFly) or concept (Payara) for managing server instances on different machines
Web Interface	Web Console	Admin Console	Browser-based tool for administering the server
Command-Line Interface (CLI)	CLI	Asadmin CLI	Tool for executing administrative commands
Programmatic Interface	HTTP Management Interface	REST Management Interface	HTTP-based API for managing the server
Individual Server Instance	Server	Standalone Instance	A single server process managed by the admin server
Configuration			
Configuration Variations	Alternative Configuration Files	Named Configurations	Different configurations for different server instances
Security			
Authentication/ Authorization	Security Domain, Realm	Realm, Login Module	Components used to verify user identities and control access to resources

Both Payara Platform Community and WildFly support the same Jakarta EE APIs, however they do it in different ways. While WildFly is mainly based on JBoss / Red Hat components, Payara Platform Community is largely built on components that are owned by the open-source Eclipse Foundation, which also owns and oversees the entire Jakarta EE specification. Nonetheless, some components used in WildFly are also used by Payara Platform Community's main application server technology, Payara Server Community, as they rely on Java EE references.

Here's the list of the most often used Jakarta EE APIs and their respective technologies in WildFly and Payara Platform Community:

Jakarta EE API	Payara Platform Community	WildFly
Contexts and Dependency Injection (CDI)	Weld	Weld
Jakarta Server Faces (JSF)	Mojarra	Mojarra
Bean Validation	Hibernate Validator	Hibernate Validator
JavaScript Object Notation (JSON) Binding	Yasson	Yasson
Jakarta Persistence (JPA)	EclipseLink	Hibernate
WebSocket	Tyrus	Undertow
RESTful Web Services (JAX-RS)	Jersey	RESTEasy
Servlet	Grizzly	Undertow
Batch	IBM 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 WildFly and Payara Platform Community provide MicroProfile APIs, but the implementation of MicroProfile APIs in Payara Server is different from WildFly. While WildFly relies on components from the SmallRye project, Payara Platform Community provides its own implementation for MicroProfile APIs.

Administration

Both WildFly and Payara Platform Community offer multiple management interfaces:

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

The Web Console in Payara Platform Community is called the Admin Console and is accessible at the 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 admin user **admin**.

The CLI for Payara Server Community is called `asadmin`. It's a script located in Payara Server installation in the **bin** directory (**asadmin** for Unix-based systems, **asadmin.bat** for Windows). This tool can manage a running instance of Payara Server Community as well as support some commands that can work without connecting to a Payara Server Community instance.

The asadmin CLI can also be used to manage starting, stopping and restarting of Payara Server instances. For example, to start Payara Server Community with the default configuration as a background service, run the `start-domain` command:

```
(syntax: Bash script)
<Payara_Home>/bin/asadmin start-domain
```

The HTTP Management Interface in Payara Server Community is mostly referred to as REST Management Interface. It's available from 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 Community contains an intuitive web client for the REST Management Interface, which is accessible via the default URL. This client can be used to explore all available management resources and commands directly from a browser. . Additionally, all management URLs support JSON and XML output for programmatic access. You can specify the desired format either by adding the format name to the path (e.g. http://localhost:4848/management/domain.json) or by specifying it using the HTTP Accept request header.

Moreover, the `asadmin` recorder is a unique feature of the Payara Platform (Community and Enterprise) that can support environment setup scripting. It records the actions performed within the Web Console and convert them as `asadmin` commands in a text file. This file can then be used to set up your server to the same state as you did manually through the Web Console.

Operating Modes

WildFly supports two operating modes:

- **Standalone:** Single server instance, operates independently, each standalone instance is configured separately. It can be used by invoking `standalone.sh`
- **Managed domain:** Designed for running and managing a multi-server topology from a single Domain Controller server. It can be used by invoking `domain.sh`

Conversely, Payara Server Community doesn't distinguish between these two modes. It always runs in a Domain mode, with a single Domain Admin Server (DAS) that can act either a standalone server, like WildFly in standalone mode, or as a central domain controller with additional managed server instances.

To use Payara Server Community as a standalone server, you simply run the DAS without adding any additional instances in the domain. A “standalone” DAS server offers both a management interface and a server that can run applications deployed to it. In this mode, all management commands automatically work on the DAS server, there’s no need to specify the command target. Alaos, applications are automatically deployed to the DAS server, any created resources and configuration changes are applied to the DAS server by default.

By default, each domain is initially configured to have a single DAS without any additional instances. Therefore, to run Payara Server Community as a standalone server, it's enough to just start the domain from the command line:

```
syntax: bash
<Payara_Home>/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 Domain Data Grid, if needed, by configuring a different grid discovery mode.

Rather than managing multiple standalone domains, you may want to leverage a multi-instance domain. To this end, a default Payara Server Community domain that has a single DAS can be extended to have one or more standalone server instances. These domain instances are managed by the 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. Also, they can automatically join the same data grid and form a high availability cluster *without* any additional configuration. Domain instances that run on the same host are grouped into a node.

On WildFly, managed instances communicate with the Domain Controller via a separate Host Controller process.

The Host Controller is a Separate Java Process:

1. **Separate from the Domain Controller:**
   - Each machine (or host) in a WildFly domain has a **Host Controller** that manages one or more **Server Instances** on that machine.
   - The **Host Controller** runs as a separate Java process on each host, distinct from the **Domain Controller**, which may run on the same machine or a different machine.
2. **Role of the Host Controller:**
   - The Host Controller's main responsibility is to communicate with the **Domain Controller** and manage the **Server Instances** on its host according to the domain-wide configuration provided by the **Domain Controller.**
   - It retrieves configuration data from the **Domain Controller** and applies it to the server instances under its control.
   - The Host Controller does not handle actual deployments or runtime activities itself—that's the job of the **Server Instances** it controls.

3. **Starting the Host Controller:**
    - The Host Controller is typically started using the domain.sh (Linux/Unix) or domain.bat (Windows) script, which is different from the standalone server script (standalone.sh or standalone.bat).
    - When started, the Host Controller process connects to the **Domain Controller** to receive its configuration instructions, which includes starting or stopping the **Server Instances** it manages.
4. **Master vs. Slave Host Controller:**
    - In a WildFly domain, there is one **Master Host Controller** that also serves as the **Domain Controller**, which centrally manages the domain configuration.
    - Other machines in the domain run **Slave Host Controllers** that connect to the Master Host Controller (Domain Controller). These Slave Host Controllers manage the server instances on their respective machines based on the configuration provided by the Domain Controller.
5. **Server Instances Controlled by the Host Controller:**
    - The **Server Instances** that run your applications (e.g., WAR or EAR artifacts) are also separate Java processes. They are started, stopped and managed by the **Host Controller.**
    - Each **Server Instance** is another JVM process running on the machine, and the Host Controller manages the lifecycle and configuration of these instances.

Domain instances within Payara Server Community communicate directly with the DAS As Payara Server doesn't require a separate Host Controller, users can benefit from highly simplified cluster setup and management.

Each standalone instance can each have its own configuration or share a common one. In both cases, configurations are managed in the DAS, which distributes configuration changes to running instances. Alternatively, instances retrieve updates from the DAS at startup.

To simplify management or create a command target, standalone instances can be organized into a **deployment group**. 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.

Please note that it's recommended that all instances in the same deployment group share the same configuration to make them behave as **replicas** of a single configuration unit. This configuration can be parameterized for each instance to account for any differences, such as specific port bindings.

A single Payara Server Community installation can contain a 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 Payara Server Community installation files. However, it is typically more useful for multiple domains to have different domain configurations. These can be switched between when needed. This is often the case during development or when running disparate test suites in different configurations.

As an example, the default Payara Server Community installation comes with one domain available out of the box: This is called "domain1" and is tuned for development purposes. Payara Server Enterprise comes with an additional domain, named "production", which is tuned for running production environments.

Thus, Payara Server Community is simpler to configure and manage than WildFly, making our solution ideal for modern containerized environments.

Clustering and High Availability

The basic concepts of clustering in Payara Server Community are very similar to those in WildFly. The following components in WildFly and Payara Server are equivalent:

WildFly	Payara Server Community	Description
Domain Controller	DAS	Main administration server
Managed Server	Standalone Instance	A server 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

Payara Server Community's Domain Data Grid is a similar concept to the Infinispan subsystem of WildFly. It provides a backbone for distributed memory and caches for instances in the Payara Server Community domain. It's flexible and can scale up and down easily just by starting or stopping additional Payara instances on the network. Unlike WildFly, which allows configuring multiple distinct cache containers, Payara Server Community provides a single Data Grid. This is pre-configured and available for all standalone instances.

Deployment Groups in Payara Server Community are comparable to Server Groups in the Managed domain mode of WildFly and offer similar functionalities. 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 them in a single step. High availability and failover mechanisms work over instances in the same deployment group.

Key Differences Between Deployment Groups and Clusters in Payara Server Community:

Feature	Deployment Groups	Clusters
Primary Focus	Simplifying application deployment	Ensuring high availability, scalability, and load balancing
Configuration Consistency	Instances can have independent configurations	Instances generally share the same configuration
High availability	Do not inherently provide high availability, instances must join a Data Grid	Provides high availability with failover and session replication
Session Replication	Not automatically provided	Session replication through Hazelcast
Use Case	Managing application deployment across heterogeneous instances	Ensuring application availability, scalability, and failover

Provisioning Support

WildFly (and JBOSS EAP) relies on a provisioning tool, called Galleon, that allows users to customize and create tailored server distributions. It is designed to help developers and administrators build custom WildFly server instances by including only the necessary components and subsystems. This capability, in turn, helps optimize server performance and reduce the overall footprint of the server.

The primary purpose of Galleon is to help reduce the footprint of WildFly by removing modules that are not required for a specific set of applications from a cluster. The Payara Platform achieves the same by providing a much smaller footprint distribution, known as Payara Micro Community. This is a lightweight, microservices-ready version of Payara Server Community. It is designed specifically for running Jakarta EE applications in microservices or cloud-native environments.

While Payara Micro Community doesn't allow for custom provisioning like Galleon, it is designed to run only the core modules needed to execute your application, thus keeping a minimal footprint. Payara Micro Community also uses a dynamic class path that helps loading only what is necessary for the application being deployed. This makes its memory footprint inherently limited compared to Payara Server Community.

Even more, Payara Server Community and Payara Micro Community support fine grained configuration of individual modules through the asadmin CLI sub-commands, which can be easily configured and executed from scripts in containerised and cloud environments.

Security

There are a number of tools that ensure the secure access to Payara Server Community and applications running on it on multiple levels:

- Transport Layer Security (TLS)/ Secure Sockets Layer (SSL) network encryption using generated self-signed and custom certificates
- Authentication using several built-in mechanisms and means to provide a custom mechanism
- Mapping of user groups to application roles
- Securing applications based on user roles
- Auditing service to record administration activity for auditing reasons

Payara Server Community comes with a pre-configured certificate and has the capability to generate additional ones as needed. Self-signed certificates are used by default for all encrypted communication, e.g. for the default HTTPS listener and the secured Java Remote Method Invocation (RMI) channel. Payara Server Community also offers multiple public certificates for known trusted certification authorities to improve the validation of chain certificates.

Authentication and authorization in Payara Server Community are provided by security realms. They are used to authenticate incoming requests using an associated Java Authentication and Authorization Service (JAAS) login module. After a login module authenticates a request, it often retrieves information about the user's roles from the realm that called it. For comparison, a security domain in WildFly can be compared to a realm in Payara Server Community.

WildFly uses realms to provide authentication and authorization mainly for securing administration management interfaces. On top of that, WildFly relies on the concept of security domains, which are used to associate multiple login modules and realms with deployed applications. For comparison, a **security domain** in WildFly can be compared to a realm in Payara Server Community, but it can combine multiple login modules.

In WildFly, login modules are not used by realms but the other way around. A WildFly 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.

Payara Server Community only allows the direct association of a single realm with an application, so that the realm takes the role of the WildFly's security domain. Multiple realms can be associated with an application using the standard Java EE Security API because Payara Server Community exposes each security realm as an Identity Store using the Payara-specific extension APIs.

The management interfaces of both WildFly and Payara Server Community are secured by a file-based mechanism. WildFly uses the **ManagementRealm** security realm, while Payara Server Community uses the **admin-realm** realm, which is an instance of a file realm.

The following table summarizes how basic security concepts in WildFly map to those in Payara Server Community:

WildFly	Payara Server Community
Security Domain	Realm
Login Module	Realm
Security Realm	FileRealm
ManagementRealm	admin-realm

Clustering in Payara Server Community and WildFly

Clustering in WildFly is composed of:

- A Domain Controller server managing several servers in a Managed Domain configuration
- Several servers and their host controllers managed by the Domain Controller
- The Infinispan subsystem that manages and coordinates distributed cache for use by WildFly clustering services

The concept of clustering in Payara Server Community is very similar to the concept of clustering in WildFly. It consists of:

- A DAS managing several Payara instances in the domain
- Several Payara instances managed by the DAS (directly, not via a host controller)
- Domain Data Grid based on Hazelcast that manages and coordinates distributed cache for use by Payara Server Community clustering services

Subsystems providing distributed memory are also similar, in their concept, to caching. Both are based on embedded in-memory data grid solutions, which are well integrated. Within WildFly, this is offered through Infinispan: 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 implementation, so that it is an implementation of the JSR-107 specification.

The same functionalities are available within Payara Server with the help of the Domain Data Grid. This is implemented using open-source Hazelcast:

- Data can be cached using a manual operation (basically, changing the statements *CacheFactory.getCache* to *hazelcastInstance.getMap* for example) or use the JCache option backed by the Hazelcast library.
- Hazelcast has support for the Java EE Transaction API specification. This means you can put data into the grid in a transactional way. It can even join in an XA transaction as part of a 2-phase commit transaction.
- Just like Infinispan, events can be fired to trigger listener code on remote nodes. Payara Server Community takes advantage of this feature for applications to trigger standard CDI events as remote events that can be observed by other standalone instances in the Domain Data Grid.

Both Infinispan and Hazelcast provide querying, distributed processing, off-heap storage and they both support cloud environments from multiple major cloud providers. Some of the more advanced features, like off-heap storage or WAN replication, are only available in Hazelcast Enterprise edition, for which a separate license is required. 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 Community.

There are also some differences between WildFly and Payara Server Community in clustering. Payara Server Community doesn't use separate Host Controller processes to manage Payara instances. Instead, the instances either just start and connect to the DAS or the DAS manages them over SSH, or Docker API directly, without any intermediary service. The DAS is even able to install and start instances this way on remote hosts without any user intervention if it has the necessary permissions via SSH. The DAS uses a concept called Node, which is a mere configuration component to store information for accessing and managing a remote host system and standalone instances installed on it.

Overview of Clustering in Payara Server Community

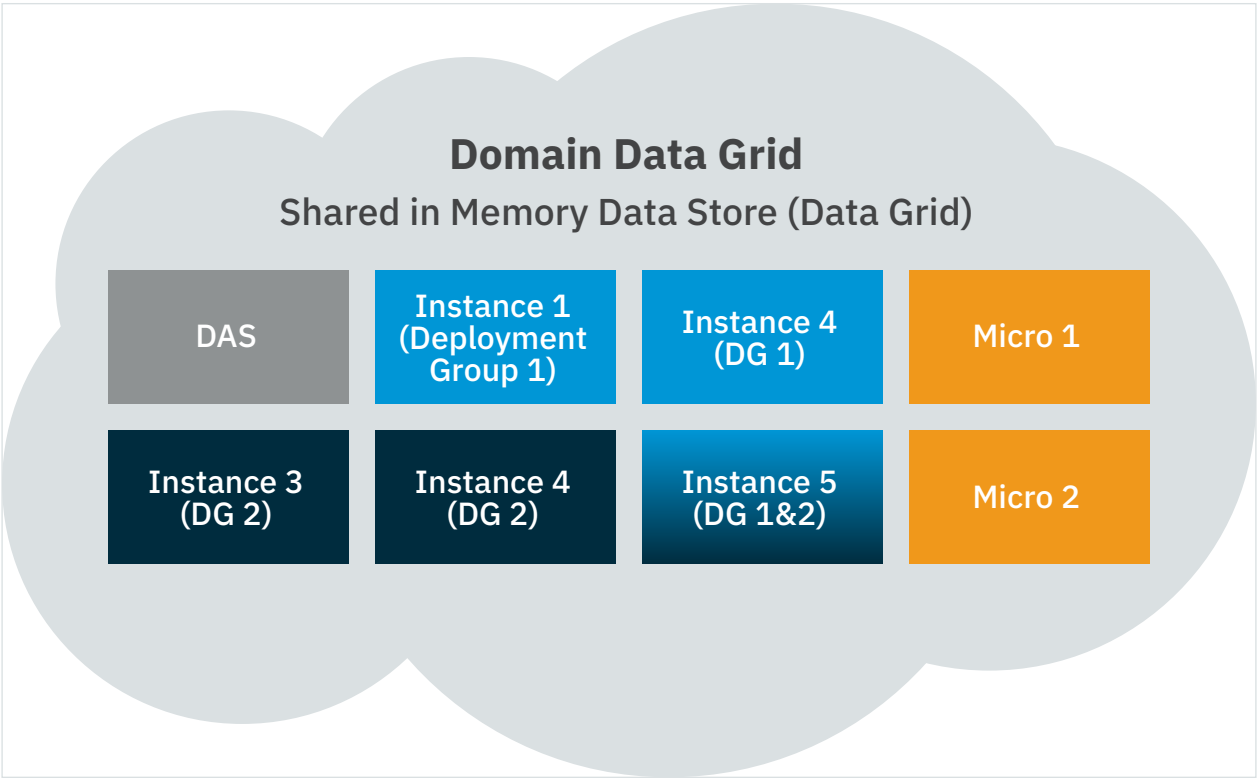
Payara Server Community supports creating cluster instances out of the box, without any additional configuration or support tools. It is possible to create a cluster instance simply by creating an additional domain instance using the DAS. The DAS and all running domain instances automatically form a data grid cluster.

Clustering in Payara Server Community has two levels. On the **first clustering level**, all running standalone instances join the same data grid, which means they participate in sharing and replicating distributed memory and caches. Payara Server Community offers several ways for standalone instances to find and join the data grid, called discovery modes (which is the concept implemented in Hazelcast). The default discovery mode, called "domain", instructs all instances to search for the DAS at a specific host and port. It's also possible to use discovery modes that support multicast,

IP address ranges and DNS or Kubernetes clusters. Payara Server Community integrates some Hazelcast configuration options natively as Payara Server Community Data Grid options. Additional configuration options can be provided by using a custom Hazelcast configuration file.

The Domain Data Grid is flexible and can scale up and down easily just by starting or stopping additional Payara instances on the network. Moreover, a single Domain Data Grid can be joined not only by Payara Server Community instances but also by Payara Micro Community instances, supporting the use of microservices.

Payara Micro Community 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 clustering 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. Also, applications and resources can be deployed or undeployed on them in a single step. High availability and failover mechanisms work over instances in the same deployment group.

Standalone 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 instances are started by the DAS:

- **SSH** - instances are started by the DAS over an SSH connection. Payara Server Community installation is copied over SSH if needed
- **DCOM** (deprecated) - instances are started by the DAS over DCOM. Payara Server Community installation is copied over DCOM if needed
- **CONFIG** - instances are not started by the DAS. They are started separately and contact the DAS via its management port
- **DOCKER** - instances are started in a Docker container using a Docker Engine REST Admin interface

High Availability Support

Payara Server Community 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 Enterprise Java Beans (EJB) data is available on all instances in the Data Grid
- Message Queue Broker connection failover - connection failover to another broker in the cluster
- Single Sign-on (SSO) state failover - SSO data is available on all instances in the Data Grid
- RMI interface over the Internet Inter-Orb Protocol (IIOP) failover, or 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 can be synchronized either by using a mechanism on top of the distributed application-level cache or via JMS brokers

Payara Server Community also supports RMI-IIOP load balancing to distribute IIOP client requests to remote EJBs evenly across a deployment group. However, the RMI-IIOP load balancing on Payara Server Community isn't based on Data Grid. 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 Community doesn't support load balancing guided by server-side load balance factors, unlike WildFly, which supports it using the `mod_cluster` subsystem.

Most of the replication and data synchronization across Payara Server Community instances uses the Data Grid powered by the community version of Hazelcast. This technology allows the flexible

connection of Payara Server Community instances to a data grid of interconnected instances, which can therefore communicate with each other and exchange data. 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.

Payara Server Community allows encrypting all data stored in the Hazelcast data grid to increase the security of the system and secure data transferred over the network.

Migrating Configuration of Server Resources

Most often, the first thing you need to migrate when porting your application from WildFly to Payara Server Community is the configuration of resources provided by the server and consumed by the application. Even if resources like JDBC data sources, mail sessions and security realms are often used by the applications in a standard and portable way, they often need to be configured in server configuration outside of the applications.

This configuration in WildFly and Payara Server Community is different and can't be just copied during migration. The following sections describe what to expect when migrating the configuration of various resources and how to configure those resources in Payara Server Community.

Data sources

Data sources are by far the most frequently used resources configured in WildFly outside of the deployed application. In WildFly, you need the following to create a data source so that it can be used by deployed applications:

1. Provide a Java Database Connectivity (JDBC) driver to WildFly
2. Define a data source that references the driver and configure its connection pool
3. Add a Java Naming and Directory Interface (JNDI) name to the data source to make it available to the applications

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

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

1. Provide a JDBC driver to Payara Server Community
2. Create a JDBC connection pool that references the driver and configure it
3. Create a JDBC source with a JNDI name that references the JDBC connection pool

In Payara Server Community, the recommended way to install the JDBC driver is to add a JAR with the driver as a server library. If you have multiple instances in a Payara Server Community domain, this JAR file will be distributed to them automatically in the same way as all custom JARs added by following the same process, so that all instances can access it.

As a result, the amount of work done in Payara Server Community to install and distribute a JDBC driver JAR is the same as WildFly. But the way how it works is different. In Payara Server Community, unlike in WildFly, the JAR won't appear in the list of deployed modules. It will be simply added to the class path for all applications to access it. This is enough for the connection pool to access and use the JDBC driver.

To work with custom JAR files in Payara Server Community, there are these `asadmin` commands:

- **`add-library`** - to install a library to the domain
- **`remove-library`** - to uninstall the library from the domain

For example, a command-line command to add a MySQL driver JAR would look like:

```
(syntax: Bash script)
<Payara_Home>/bin/asadmin add-library /path/to/db-driver.jar
```

After you install the driver JAR, you can continue by creating a JDBC connection pool that uses the driver, followed by creating a JDBC resource that exposes the connection pool as a data source via JNDI.

As an example, to create a data source for H2 database using the asadmin CLI, you can use the following command:

```
create-jdbc-connection-pool --datasourceclassname org.h2.Driver --restype
javax.sql.XADataSource --property portNumber=1527:password=APP:user=APP:serverN
ame=localhost:databaseName=sun-appserv-samples sample_h2_pool
```

Security Realms

Enterprise Java and Jakarta EE applications are often secured by a mechanism specified by the realm name in the application descriptors. However, the application only specifies the name of the realm to use, and all the configuration needs to be done in the server.

In WildFly, the realm name specified in the application 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 WildFly. The resulting security chain during authentication is as follows:

1. application is associated with a single security domain
2. the security domain delegates to one or more login modules
3. a login module authenticates the user and optionally provides a list of roles
4. if used, RealmDirect login module delegates to a realm to authenticate and provide a list of roles

Realms in Payara Server Community are different than realms in WildFly, as they behave in a similar manner to WildFly login modules. Unlike WildFly, which allows connecting multiple login modules to a security domain, *Payara Server Community binds the application's realm name directly to a realm on the server.* While this can reduce flexibility because *only a single realm can be used with an application*, it can also simplify the configuration.

The chosen realm then delegates authentication and authorization to a JAAS login module bound by its name (JAAS context). Therefore, the security chain in Payara Server Community is:

1. application is associated with a single realm
2. the realm usually delegates to a JAAS login module for authentication
3. optionally, the login module retrieves the list of roles from the realm

If a more complex authentication and authorization mechanism is required, a custom realm needs to be developed and installed into Payara Server Community. Alternatively, multiple realms can be associated with an application using the standard Java EE Security API.

If a security domain in WildFly uses only a single login module that matches a realm in Payara Server Community, then the migration to Payara Server Community is rather straightforward, as the login module can be directly converted to a matching realm. This, in turn, will be given the same name as the security domain in WildFly so that it matches the deployed application.

As an example, consider the following login module that uses an H2 data source from WildFly:

```
<security-domain name="mySecurityDomain" cache-type="default">
    <authentication>
        <login-module code="Database" flag="required">
            <module-option name="dsJndiName" value="java:/jdbc/MyDataSource"/>
            <module-option name="principalsQuery" value="SELECT password FROM
users WHERE username = ?"/>
            <module-option name="rolesQuery" value="SELECT role, 'Roles' FROM
roles WHERE username = ?"/>
            <module-option name="hashAlgorithm" value="SHA-256"/>
        </login-module>
    </authentication>
</security-domain>
```

This module can be changed into a Payara Server Community realm by first creating a data source like the following:

```
<security-domain name="mySecurityDomain" cache-type="default">
    <authentication>
        <login-module code="Database" flag="required">
            <module-option name="dsJndiName" value="java:/jdbc/
MyDataSource"/>
            <module-option name="principalsQuery" value="SELECT password
FROM users WHERE username = ?"/>
            <module-option name="rolesQuery" value="SELECT role, 'Roles'
FROM roles WHERE username = ?"/>
            <module-option name="hashAlgorithm" value="SHA-256"/>
        </login-module>
    </authentication>
</security-domain>
```

And then using the data source within a JDBC realm, which would look something like the following:

```
<security-service>
    <auth-realm name="JDBCRealm" classname="com.sun.enterprise.security.auth.
realm.jdbc.JDBCRealm">
        <property name="jaas-context" value="jdbcRealm"/>
        <property name="datasource-jndi" value="jdbc/MyDataSource"/>
        <property name="user-table" value="users"/>
        <property name="user-name-column" value="username"/>
        <property name="password-column" value="password"/>
        <property name="group-table" value="roles"/>
        <property name="group-name-column" value="role"/>
        <property name="digestrealm-password-enc-algorithm" value="SHA-256"/>
    </auth-realm>
</security-service>
```

However, there may be differences in how some matching realms are configured and behave. The following table compares realms in Payara Server Community to similar login modules in WildFly:

WildFly Login Module	Payara Server Community Realm	Notes
Certificate, CertificateUsers	CertificateRealm	
CertificateRoles	CertificateRealm	WildFly reads roles from a file. Payara Server Community uses mapping rules in application descriptors.
Database, DatabaseUsers	JDBCRealm	WildFly uses a query to retrieve users and roles. Payara Server Community expects a certain database structure.
Ldap, LdapUsers	LDAPRealm	
Simple, PropertiesUsers, UsersRoles, RealmDirect	FileRealm	FileRealm stores usernames, passwords and roles in a file with a custom format, not as properties files.

JavaMail Sessions

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

1. connection and authentication to the Simple Mail Transfer Protocol (SMTP) server
2. JNDI name of the resource
3. default sender address

This all can be done using a single `asadmin` command called **`create-javamail-resource`**. For example:

```
asadmin create-javamail-resource \
    --mailhost smtp.example.com \
    --mailuser user@example.com \
    --fromaddress user@example.com \
    --enabled true \
    --storeprotocol imap \
    --storeprotocolclass com.sun.mail.imap.IMAPStore \
    --transportprotocol smtp \
    --transportprotocolclass com.sun.mail.smtp.SMTPTransport \
    --property mail.smtp.auth=true:mail.smtp.starttls.enable=true:mail.smtp.
port=465:mail.smtp.ssl.enable=true \
    mail/Default
```

Conversely, WildFly splits the configuration to two components and first requires the creation of an outbound socket binding for the SMTP connection. Subsequently, it is necessary to create the JavaMail session resource with a JNDI name and this binding.

A typical WildFly configuration looks like the following:

```
<subsystem xmlns="urn:jboss:domain:mail:3.0">
    <mail-session jndi-name="java:/jboss/mail/Default" debug="false">
        <smtp-server outbound-socket-binding-ref="mail-smtp">
            <login name="user@example.com" password="password"/>
            <ssl enabled="true"/>
        </smtp-server>
    </mail-session>
</subsystem>

<outbound-socket-binding name="mail-smtp">
    <remote-destination host="smtp.example.com" port="465"/>
</outbound-socket-binding>
```

Such a configuration can be mapped to the following Payara Server Community configuration:

```
<resources>
    <mail-resource
        jndi-name="mail/Default"
        mail-host="smtp.example.com"
        mail-user="user@example.com"
        from="user@example.com"
        mail-password="password"
        enabled="true">
        <property name="mail.smtp.auth" value="true"/>
        <property name="mail.smtp.starttls.enable" value="true"/>
        <property name="mail.smtp.port" value="465"/>
        <property name="mail.smtp.ssl.enable" value="true"/>
    </mail-resource>
</resources>
```

Migrating Keycloak

Applications on WildFly are often secured using Keycloak, which supports centralized Identity and Access Management and Single Sign-On. From Keycloak 17, the default Keycloak distribution is powered by Quarkus. Legacy WildFly powered distribution was supported only until June 2022. Payara Server Community supports the connection to a standalone instance of Keycloak. Hence, there are several alternatives for migrating your security infrastructure to Payara Server Community:

- Easily configure Payara Server Community to use your existing Keycloak server via OpenID Connect (OIDC) protocol.
- Use the native Single Sign-On solution in Payara Server Community
- Migrate from Keycloak to another Identity and Access Management solution

The best way to integrate with Keycloak is to migrate your application to authenticate using OAuth 2.0 and use Keycloak as an OAuth 2.0 provider. Keycloak can easily be run as a standalone server in a docker container or in the cloud. OAuth 2.0 is supported in Payara Server Community using the OAuth 2.0 authentication definition in combination with the standard Jakarta Security API via the @ OpenIdAuthenticationMechanismDefinition annotation. Within your application code, you can easily get the Role and Principal details by injecting the OpenIdContext Object with the following code:

```
@Inject
      OpenIdContext openIdContext;
openIdContext.getClaims();
```

It is also possible to authenticate on the web page using an OAuth 2.0 JavaScript library and pass the acquired JWT token to the backend services using the MicroProfile JWT mechanism supported by Payara Server Community.

You can read more about OAuth 2.0 authentication definition in Payara Server Community here and about MicroProfile JWT here.

Payara Server Community also provides native SSO support. This can easily replace Keycloak if all applications are deployed in the same Payara Server Community domain and they authenticate users through the same security realm. When SSO is enabled on Payara Server Community, a user that is authenticated by one application is then automatically authenticated in all other applications that use the same realm until logged out in one of the applications.

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 Community, e.g. Lightweight Directory Access Protocol (LDAP), can be used to authenticate users.

Cloud

Containerized environments are becoming more important these days. WildFly has its own official example Docker image, and various examples can be found on how an Orchestration tool like Kubernetes can be used in combination with WildFly.

Payara Server Community is no exception in the cloud evolution. We too have official Docker images on DockerHub. Also, we created Payara Micro Community, a specialized packaging of Payara Server Community that is optimized for cloud and micro-service environments. With Payara Micro Community, you can easily create a setup that is maintained by orchestration tools, such as Kubernetes and Docker Swarm, and even makes use of the functionality of these tools, like DNS management within Kubernetes.

Payara Services also provides Docker images for Payara Server Full Community and Payara Server Node Community, which can help you to form a traditional domain managed cluster within containerized environments. You can find the images here: https://hub.docker.com/u/payara. As a result, just like on WildFly, you can run Payara Server Community on cloud providers like Azure, Amazon, Google Cloud, as well as on OpenShift.

IDE Support

All major Integrated Development Environments (IDEs), like IntelliJ® IDEA, Apache NetBeans, Eclipse® IDE and Visual Studio Code, are supported by both servers. Thus, you can use your favourite IDE to develop your application. Apache NetBeans supports Payara Server Community out of the box. The plugins for the other IDEs are maintained by the Payara Service team.

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

Payara Server Community also supports automatic deployment and redeployment of an application by dropping the application artifact in the *auto-deployment* folder. This can be useful if no IDE can be used for automatic redeployment.

Innovation

The team behind Payara Server Community is also actively working on the future of enterprise Java 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 our application server technology was one of the first that combined the MicroProfile implementations and the ability to run Jakarta EE applications in a single runtime. This gives you the ultimate flexibility to choose the right mix of dependencies for your use case while allowing you to implement a strategy of gradual migration from the Java EE platform to a more micro-service alike application structure.

For microservice applications, Payara Community provides Jakarta EE Web Profile specifications, some additional Jakarta EE specifications, like Jakarta Concurrency, as well as 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 runtime. This reduces image sizes, bandwidth usages and deployment times.

But enhancements are not only implemented outside the Java EE area. For example, Payara Server Community has many additions in the area of application security. Using the Security API (added in Java EE 8), developers can benefit from various authentication and authorization schemes within Payara Platform Community, such asOAuth2, OpenIdConnect and JWT tokens.

Why Payara Platform Community?

Payara Platform Community is notably better then WildFly in the following areas ：

Cloud-Native and Aggressively Compatible

Payara Platform Community is optimized for cloud, on-premises and hybrid environments. As Payara Services is a Solutions Member of the Eclipse Foundation and Strategic Member of the Jakarta EE working group, you'll find that Payara Platform Community is positioned for future compliance with Jakarta EE.

Besides, Payara Platform Community is container-friendly, incorporating Docker and Kubernetes in its offering. It is also compatible with services you may already be using, such as Microsoft Azure™, Amazon AWS and MicroProfile.

User-Friendly and Intuitive

Payara Platform Community offers a comprehensive and user-friendly administration console with intuitive navigation, making it ideal for teams of varying expertise. Its straightforward setup and configuration reduce onboarding time, maximizing developers' productivity.

Open-Source Software with a Future You Help Define

Being open-source, Payara Platform Community allows you to submit your ideas, feedback and collaboration to ensure Payara Server Community is the best option for developing, experimenting and testing enterprise Java applications, and Payara Micro Community the best option for enterprise Java applications in a modern virtualized infrastructure.

In addition, Payara Platform Enterprise customers are invited to customer advisory calls every six months to help drive the evolution of Payara Platform, as new features and enhancements are developed to meet customer needs.

Stable, Production-Ready Option With Full Support

With seamless upgrade paths to the commercial Payara Platform Enterprise, Payara Platform Community provides flexibility for growth as organizational needs evolve. When you download the Payara Platform Enterprise, you're downloading and adopting a production-ready version of Payara Server or Payara Micro that is suitable for mission-critical applications. You can download a production-ready version from the download section of the website.

With a 10-year support lifecycle and a monthly release schedule for Payara Platform Enterprise customers that includes bug fixes and patches, Payara Platform Enterprise offers security and stability without the need for upgrading every year or two. Payara Platform Enterprise customers enjoy fast issue resolution directly from our global team of expert engineers for both production and development issues. Users can also benefit from prioritized access to new features and send specific feature requests.

sales@payara.fish

UK: +44 800 538 5490
Intl: +1 888 239 8941

www.payara.fish