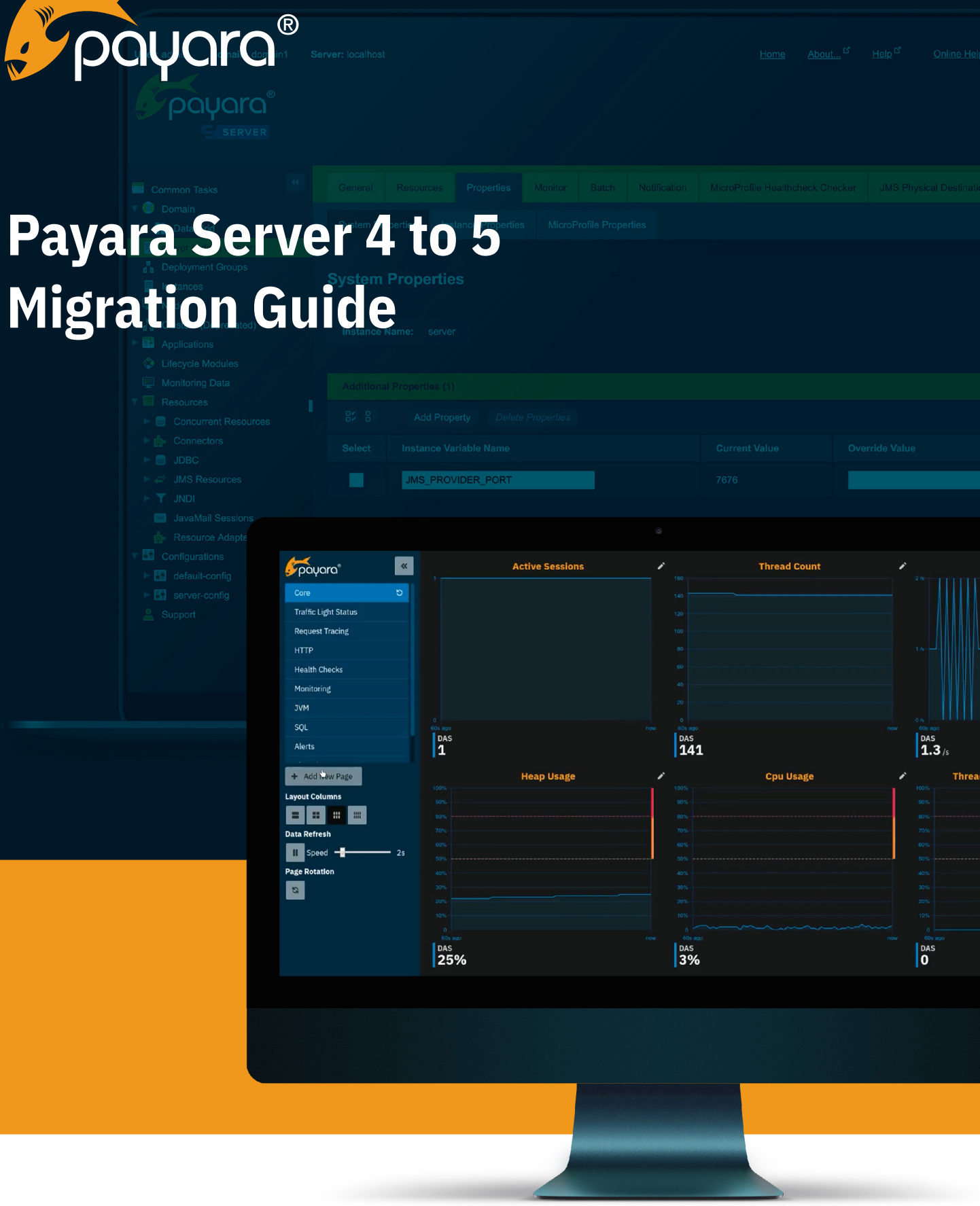
Payara Server 4 to 5 Migration Guide



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

User Guide

Contents

Introduction

Payara Server 5 was introduced at the beginning of 2018 as the next major version of Payara Server. Payara Server 4 was derived from the GlassFish Open Source Edition 4.1 and was the recommended option for many users looking for a drop-in replacement for their GlassFish server installations. Constant feedback from our customers and the community about the user experience of our product has led our evolution of Payara Server to meet user expectations. We have concluded that, although Payara Server 4 is a reliable option in the market for both developers and operation staff, there is room to implement improvements and changes to leverage the productivity levels required by the current environment. Thus Payara Server 5, which is derived from GlassFish Server Open Source Edition 5, will deviate some of its features and internal mechanisms from the ones implemented on GlassFish to offer the productivity and functionality that our user base really needs. As such, Payara Server 5 targets the following goals:

- Cloud and container friendly
- Compatible with modern Java APIs
- Reduce the dependency on legacy components and/or third-party libraries
- Improve the general performance and quality of deployed applications
- Provide top-level security and monitoring features

The purpose of this guide is to help you prepare and understand the main challenges you may face when migrating from Payara Server 4 to Payara Server 5. Be sure to understand this entire document before planning the migration in your projects to have a better understanding of which steps to take in your particular case. Payara Server Enterprise or Payara Micro Enterprise customers can submit support tickets with migration questions for assistance, or should you want more hands-on guidance through the migration process, take a look at our upgrade service as part of our Payara Accelerator consultancy. If you're using the community version of Payara Server or Payara Micro, we invite you to take a look at our production-ready, fully supported options, Payara Enterprise, which gives you access to a choice of support included in your subscription: Migration & Project Support, 24x7, 10x5.

Main Advantages of Payara Server 5

Clustering and High Availability Improvements

High availability is a concept familiar to most developers and server administrators. For mission-critical or high-performance applications and services it is imperative to coordinate a high-availability strategy so that the business is not affected in case of failure or high load. Payara Server comes equipped with the concept of a Domain Data Grid which has the following responsibilities:

- Share the data across all of the instances in the domain and replicate such data in case of fail-over
- Provide a centralized configuration for all instances in the domain

On top of the Domain Data Grid, applications and resources can be assigned to multiple groups called Deployment Groups. These provide the following features:

- Function as a deployment “target”, meaning applications and resources deployed to a deployment group are deployed automatically to all instances in the group
- Allow controlling of multiple instances in the domain with a single action (e.g. start/stop all instances in the group

This leads to better integration of Hazelcast into high availability services and easier configuration of the Hazelcast based cluster. It also provides a lot more flexible clustering options; it enables an easy way to combine options for dynamic formation of a cluster suitable in scalable environments with allowing more control over instances and deployments via Deployment Groups.

The Domain Data Grid can be compared to clustering with Hazelcast in Payara Server 4, and Deployment Groups are similar to clusters in Payara Server 4. However, Deployment Groups are built on top of the Domain Data Grid and thus are powered by Hazelcast, unlike clusters in Payara Server 4, which are based on a technology called Shoal (or GMS) which has been completely removed in Payara Server 5. The configuration and administration commands for clusters in Payara Server 4 are still supported by Payara Server 5 so that it’s easy to migrate them, but clusters in Payara Server 5 created and managed this way run on the same technology as Domain Data Grid and behave as any other Deployment Group as the older Shoal Clusters are deprecated and not used by default for new clustering configurations. Deployment Groups and their associated administration commands provide a complete replacement.

Cloud Deployment Improvements

One of the main disadvantages of Payara Server 4 is that while Hazelcast clustering features are provided in addition to the traditional clustering model provided by Shoal, the Hazelcast clustering features are not user-friendly to use in common cloud deployment scenarios, especially in environments where container technologies form the backbone of the topology (like Docker or Kubernetes, for example). Payara Server 5 includes several improvements in the form of better clustering integration with cloud environments and friendlier configuration options that cover most common use cases in cloud environments. Example of discovery modes provided by the Domain Data Grid include:

- **TCPIP:** Discovering instances that live in a list of hosts identified by their IPv4 or IPv6 network addresses
- **DNS:** Discovering instances that live in a list of hosts identified by host name
- **Multicast:** Allowing instances to "talk" to the cluster by using the multicast protocol and join it themselves
- **Kubernetes:** Discovering instances that live in hosts within a Kubernetes cluster

One important feature to discuss when mentioning cloud environments is elasticity, which is the capability of a cloud-environment to scale-up or down depending on the expected user load (and other factors). Elasticity is one of the main draws of most cloud environments, and the Domain Data Grid is equipped to allow elastic arrangements on most of these cloud environments. An important thing to consider when developing an elastic arrangement with the Payara Platform, is that by default both Payara Server and Payara Micro support **elastic clustering** via the Domain Data Grid.

Payara Server also supports grouped deployments but grouped deployments do not support **elasticity** since a deployment group targets a specific set of instances that have to be centrally configured. Such deployment groups are therefore more suitable as a replacement for a traditional centrally managed clustering (more information on how to use the Domain Data Grid and Deployment Groups will be provided in the following sections). Payara Micro on the other hand does not support the deployment group concept; the life cycle of any deployed application is tied to the life cycle of the instance itself. This is a design choice because Payara Micro is built specifically for elastic cloud environments.

Migration Process

Preparation

Payara Server 5 **requires the use of JDK 8 at the very least**. If your Payara Server 4 domains are currently running on JDK 7, you will have to update your JDK installations before starting the migration.

You also have to keep in mind that:

- Payara Server 5 supports Java/Jakarta EE 8 applications. When migrating your Payara Server 4 installation, you must be careful if your application uses JSON serialization of Java objects, since Java EE 8 includes the new JSON-B API which might break your existing applications. Additionally, there is a new iteration of the Servlet API (4.0) that introduces HTTP/2 support. More information about these two topics are explained in the following sections.
- Payara Server 5 also supports MicroProfile, however, the MicroProfile APIs are in constant change so the version of Payara Server 5 that you are migrating may use a newer version of one of its APIs (like Metrics), which can introduce breaking changes. It is recommended that you check which APIs are affected and refactor the application's code to use the newer APIs.

Migrating a Domain from Payara Server 4 using Backup and Restore

One of the recommended strategies that you can use to migrate your working Payara Server 4 domain to Payara Server 5 is execute a backup of this domain and then restore it under Payara Server 5. Keep in mind that this strategy will import your current configuration as it is into Payara Server 5, so in order to use the new features included (like the Domain Data Grid, H2 database, HTTP/2 protocol, etc.) you will have to implement specific configuration changes mentioned in the following sections.

Follow these steps to implement this strategy on your environment:

1. First, you need to stop the running Payara Server 4 domain. The domain backup process will only work when the domain in question is not running, so you will have to schedule a period of downtime for your current Payara Server 4 production domain.
2. Run the `backup-domain` asadmin command and specify the path to a directory where a compressed file holding the domain backup will be stored:

```
asadmin> backup-domain --backupDir <path-to-backup-directory> <domain-name>
```

The resulting compressed file will then be stored in this location:

```
<path-to-backup-directory>/<domain-name>/<domain-name>_<yyyy_mm_dd>_v<backup_
number>.zip
```

Where the **backup_number** placeholder represents a consecutive integer that counts the current number of backup operations executed on the domain.

3. With the domain fully backed-up, you can now proceed to restore it under your new Payara Server 5 installation. Run the following command:

```
asadmin> restore-domain --filename <path-to-backup-directory>/<domain-
name>/<domain-name>_<yyyy_mm_dd>_v<backup_number>.zip --long <domain-name>
```

The command should print out a detailed report of the restoration outcome:

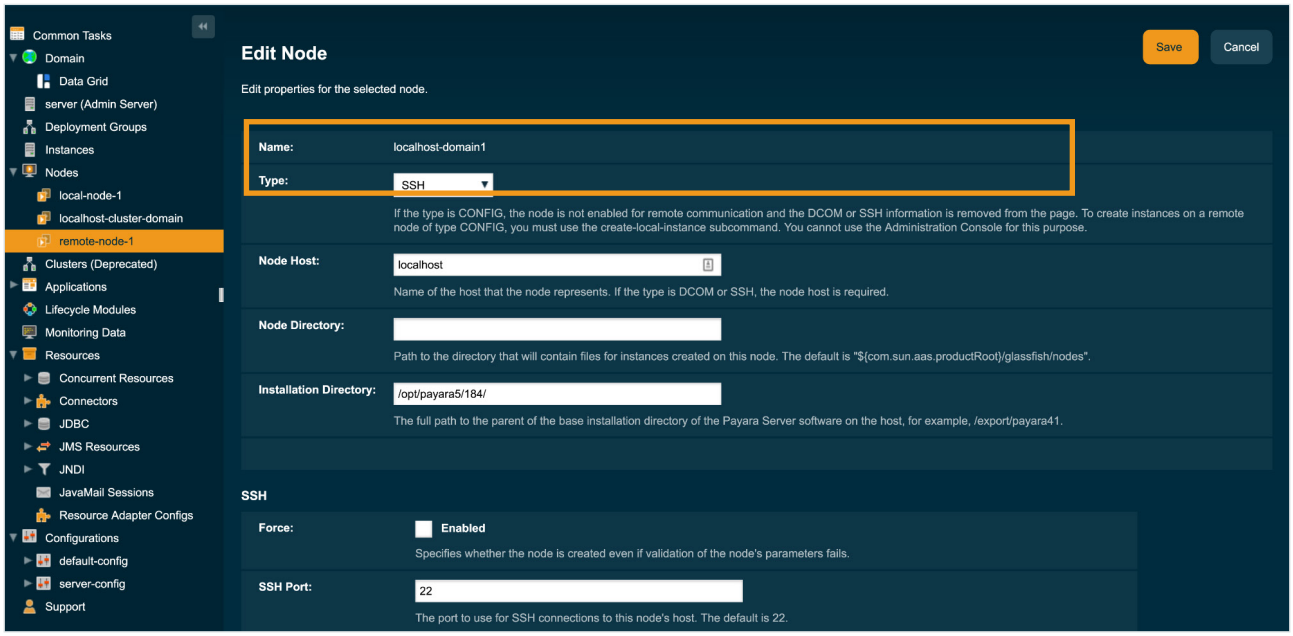
```
Restored the domain (<domain-name>) to /opt/payara5/184/glassfish/
domains/<domain-name>
Description                : <domain-name> backup created on <yyyy_mm_dd> by
user <username>
GlassFish Version          : Payara Server 5.184 #badassfish (build 24)
Backup User                : <username>
Backup Date                : <backup-timestamp>
Domain Name                : <domain-name>
Backup Type                : full
Backup Config Name         :
Backup Filename (origin)   : <path-to-backup-directory>/<domain-name>/<domain-
name>_<yyyy_mm_dd>_v<backup_number>.zip
Domain Directory           : /opt/payara5/184/glassfish/domains/<domain-name>

Command restore-domain executed successfully.
```

4. And finally, proceed to start the restored domain!

Additional Considerations for Nodes and Instances

If your domain configuration includes the definitions of instances that live in separate nodes, you must consider the following set of recommendations in order for the domain to be fully workable when restored:

1. You will have to do a manual installation of the Payara Server 5 binary files in the same locations as they are defined in the domain configuration. Keep in mind that you will have to replace the Payara Server 4 binaries in that case, which means that you must forego your working Payara Server 4 domain. This could be a problem if you want to revert your Payara Server 5 installation and go back to Payara Server 4, so to prevent that it's best that you change the installation directory in the remote hosts. Change this configuration on the Admin Console before starting the remote instances:



2. You will have to re-synchronize the creation and association of these instances to the DAS. In order to do this, you will have to manually start the instances in each of the nodes (be them local or remote nodes) configured within the domain. When starting these instances, set the --sync argument to full so that each instance is re-created successfully:

```
asadmin> start-local-instance --sync=full <instance-name>
```

Special Considerations for Payara Server 5.201

Payara Server 5.x, in its release 5.184 introduced a set of specific requirements on the JDK 8 update needed to run the server were included out of necessity in order to circumvent changes needed by several SSL related classes that are included with the Grizzly NPN framework. This framework provisions the HTTP/2 protocol for the Web Container, and depending on the version of the framework, there are exact requirements for the version of JDK being used as well. If an incompatible JDK 8 update is being used with Payara Server 5, the server's startup will be affected. The best way to solve this (and future compatibility issues) is to manually update the domain configuration of the server:

1. Open your domain configuration file (***domain.xml***) and locate the following JVM argument setting in the `server-config` configuration tree:

```
<java-config classpath-suffix="" debug-options="-agentlib:jdwp=transport=dt_
socket,server=y,suspend=n,address=9009" system-classpath="">
   <!-- ... -->
   <jvm-options>-Xbootclasspath/p:${com.sun.aas.installRoot}/lib/grizzly-npn-
bootstrap.jar</jvm-options>
   <!-- ... -->
</java-config>
```

2. Replace the JVM setting with the following new set of elements:

```
<java-config classpath-suffix="" debug-options="-agentlib:jdwp=transport=dt_
socket,server=y,suspend=n,address=9009" system-classpath="">
   <!-- ... -->
   <jvm-options>[1.8.0|1.8.0u120]-Xbootclasspath/p:${com.sun.aas.installRoot}/
lib/grizzly-npn-bootstrap-1.6.jar</jvm-options>
   <jvm-options>[1.8.0u121|1.8.0u160]-Xbootclasspath/p:${com.sun.aas.
installRoot}/lib/grizzly-npn-bootstrap-1.7.jar</jvm-options>
   <jvm-options>[1.8.0u161|1.8.0u190]-Xbootclasspath/p:${com.sun.aas.
installRoot}/lib/grizzly-npn-bootstrap-1.8.jar</jvm-options>
   <jvm-options>[1.8.0u191|1.8.0u500]-Xbootclasspath/p:${com.sun.aas.
installRoot}/lib/grizzly-npn-bootstrap-1.8.1.jar</jvm-options>
   <jvm-options>[9|]-Xbootclasspath/a:${com.sun.aas.installRoot}/lib/grizzly-
npn-api.jar</jvm-options>
   <!-- ... -->
</java-config>
```

With that, your migrated domain should be compatible with the corresponding JDK 8 update, and the domain will be ready for future migrations as well. Additionally, when considering upgrading to JDK 11, the domain will be prepared as well to run with the correct Grizzly Bootstrap NPN API version.

> ***If your domain has multiple configurations that are used for running additional instances, you must apply the same changes in their configuration trees as well.***

Clustering and High-Availability

Summary of Clustering Options in Payara Server 4

Payara Server 4 supports two mechanisms of clustering. Payara Server 5 improves the one based on Hazelcast and integrates it better into the domain configuration. On the other hand, while the traditional clustering inherited from GlassFish is still present, it's now deprecated.

The first mechanism supported by Payara Server 4 is based on the (already deprecated) Shoal project, which is the traditional clustering mechanism which Payara Server inherited from GlassFish Open Source Server. A Shoal cluster needs to be prepared in a systematic manner and new instances can be manually added or removed as well. Although this mechanism is reliable, there are is set of multiple limitations that have piled up over the years:

- Preparing a cluster requires many things: setting up the cluster in the DAS, setting up each of the nodes either local or remote, setting up SSH access across all cluster hosts (in the case of remote nodes)
- Since instances must be added or removed manually to the cluster, in a cloud environment, scaling up or down is usually a cumbersome and extremely tedious task
- Only specific data (web session data and Stateful Session Beans) is replicated and stored across the cluster
- The protocol internals used for establishing the communication across instances haven't aged well with the side-effect of performance degradation over the lifetime of the cluster

The second mechanism is Hazelcast Clustering, which uses Hazelcast to configure a customized cluster that allows both new Payara Server and Payara Micro instances to automatically join and leave the cluster when necessary. This functionality was made available starting with Payara Server 4.1.1.161 and was created to make Payara Server friendlier with cloud environments and simplify the provisioning work needed to quickly set-up a high-availability environment. However, with these benefits another set of challenges was introduced as well:

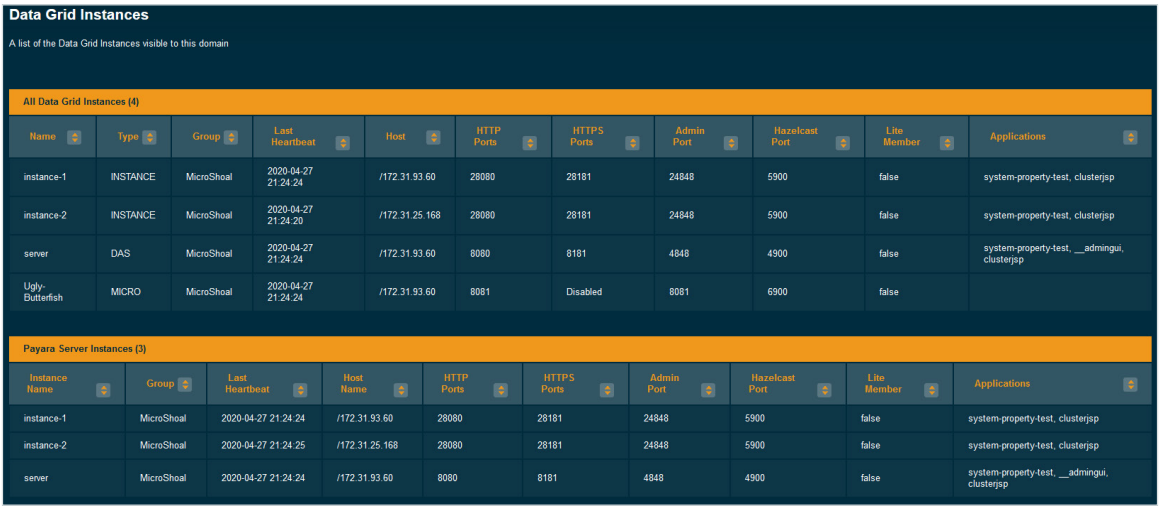
- Hazelcast must be enabled manually in all nodes/instances of the server
- Although Hazelcast allows auto-discovery of new instances, allowing them to automatically join a cluster when detected, in specific environments when multiple clusters must be provisioned, the setup can be cumbersome because -
- Hazelcast by default uses the multicast protocol to communicate to all nodes across the cluster. Some cloud providers and container orchestration tools do not support the multicast protocol, so, a customized Hazelcast configuration file must be used to provision all cluster instances to use another protocol. This is a common occurrence.

Domain Data Grid in Payara Server 5

To overcome all clustering challenges in Payara Server 4, Payara Server 5 introduces the concept of **Domain Data Grid**. The Domain Data Grid provides an in-memory data structure that is distributed amongst all Payara Server instances within a Payara Domain. The Data Grid is highly available, highly scalable, and enables in-memory data storage and replication among all Payara Server instances in a domain.

The Domain Data Grid is an evolution of the Hazelcast Clustering supported in Payara Server 4. Upgrading from this type of clustering to Domain Data Grid is seamless and doesn't require any configuration changes. If you use Shoal Clustering in Payara Server 4, you can continue managing the same clusters in Payara Server 5 with the same administration commands, however they won't use the underlying Shoal/GMS technology but instead will run with the assistance of Hazelcast and behave in a similar manner to Deployment Groups. You should pay closer attention to the section about clustering changes further in this guide and plan the corresponding migration to Domain Data Grid or to the Deployment Groups add-on.

In Payara Server 5, **Hazelcast is enabled by default** compared to Payara Server 4 where it had to be enabled manually (Enabling Hazelcast is also a requirement to use other features like the JCache API or using Hazelcast as a data store for *Web Sessions Persistence* for example). This means that, by default, **all instances in the domain will automatically join the Domain Data Grid** and benefit from its features, including session replication, distributed caches and an embedded Hazelcast grid. The Domain Administration Server (DAS) will know all instances and coordinate all corresponding communication between them. The DAS can also display information about all instances in the Domain Data Grid either by using `asadmin` commands or the Admin Console as on the following picture:



Data Grid Instances

A list of the Data Grid Instances visible to this domain

All Data Grid Instances (4)

Name	Type	Group	Last Heartbeat	Host	HTTP Ports	HTTPS Ports	Admin Port	Hazelcast Port	Lite Member	Applications
instance-1	INSTANCE	MicroShoal	2020-04-27 21:24:24	/172.31.93.60	28080	28181	24848	5900	false	system-property-test, clusterjsp
instance-2	INSTANCE	MicroShoal	2020-04-27 21:24:20	/172.31.25.168	28080	28181	24848	5900	false	system-property-test, clusterjsp
server	DAS	MicroShoal	2020-04-27 21:24:24	/172.31.93.60	8080	8181	4848	4900	false	system-property-test, __admingui, clusterjsp
Ugly-Butterfish	MICRO	MicroShoal	2020-04-27 21:24:24	/172.31.93.60	8081	Disabled	8081	6900	false	

Payara Server Instances (3)

Instance Name	Group	Last Heartbeat	Host Name	HTTP Ports	HTTPS Ports	Admin Port	Hazelcast Port	Lite Member	Applications
instance-1	MicroShoal	2020-04-27 21:24:24	/172.31.93.60	28080	28181	24848	5900	false	system-property-test, clusterjsp
instance-2	MicroShoal	2020-04-27 21:24:25	/172.31.25.168	28080	28181	24848	5900	false	system-property-test, clusterjsp
server	MicroShoal	2020-04-27 21:24:24	/172.31.93.60	8080	8181	4848	4900	false	system-property-test, __admingui, clusterjsp

The Domain Data Grid will be **composed of running instances only**. By default, all instances created in a domain join the data grid when started. The Domain Data Grid can also contain instances that aren't configured in the domain if they are configured to connect to the same data grid.

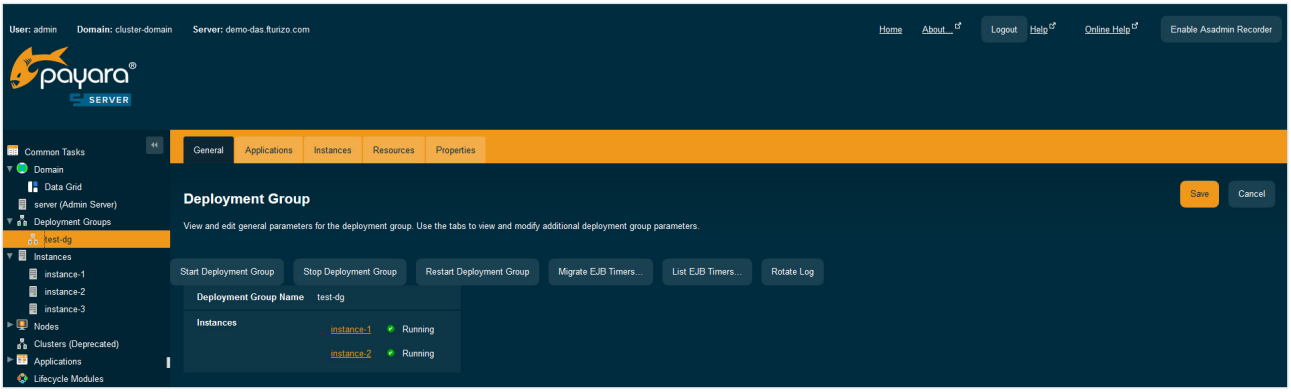
The instances that join a data grid are categorized in the following types:

- **DAS:** The Domain Administration Server itself
- **INSTANCE:** Individual instances that are part of the same domain as the DAS or are from a separate domain)
- **MICRO:** Payara *Micro* instances that explicitly connect to the domain grid.

Deployment Groups in Payara Server 5

While the Domain Data Grid is very flexible, only a single grid exists within the domain. All resources associated to a Domain Data Grid are shared by all instances in the data grid. Moreover, each instance in the data grid is managed separately and applications are also deployed separately to each instance. This is completely acceptable, sometimes even desirable in a dynamic scalable environment. However, the Domain Data Grid itself doesn't provide all of the features of the old Shoal clustering model. That's why Payara Server 5 introduces the concept of *Deployment Groups*.

Deployment Groups work as an extension to the Domain Data Grid functionality: A deployment group is a managed collection of instances that share the same applications and resources. This collection of instances can provide load-balancing and fail-over functionality as an extension to the Domain Data Grid, effectively making them work in a similar vein to old Shoal clusters.



You can see above an example of a deployment group configuration. While instances `instance-1` and `instance-2` are in the deployment group called `test-dg`, a third instance called `instance-3` is not part of it and needs to be managed separately.

Standalone Instances

Payara Server 4 has a specific distinction for two types of instances:

- *Cluster Instances,* which are the instances created directly under a cluster and are exclusive to each cluster and their life cycle is tied to that of the cluster directly.
- *Standalone Instances,* which are the instances that do not belong to a cluster. Standalone instances are completely isolated from within each other, which means that they do not share resources nor applications. Each standalone instance must be managed separately.

In Payara Server 5 however, there is no explicit distinction for instances regarding the context of the Domain Data Grid and Deployment Groups. All instances created under this model are treated effectively as standalone instances for the purposes of management and administration. This means that the same administration commands that manage an instance life cycle (`create-instance`, `delete-instance`, etc.) in Payara Server 4 will work in the same manner on Payara Server 5. The main distinction is that instances on Payara Server 5 will automatically join the Domain Data Grid and *can* be added to Deployment Groups. On Payara Server 4, standalone instances *can't* be added to traditional Shoal cluster but they can join a Hazelcast cluster if their configuration is modified. *Cluster instances* in Payara Server 5 still exist as part of the old Clustering Model that is only present for legacy purposes and they behave very similar to other standalone instances grouped in a deployment group.

This distinction must be clarified in case your GlassFish has standalone instances. When migrating your domain to Payara Server 5, these instances will still work correctly but will join the Domain Data Grid automatically, They will provide space for the shared replicated memory unless they are configured as **lite instances**, which don't provide storage for the shared memory.

Lite instances of Domain Data Grid are instances, which don't keep any shared data in their heap but still can access shared memory which is available on other instances in the grid. Lite instances are part of the data grid as all other instances, have access to the shared memory and all other grid features as all other instances. You can turn any existing standalone instance into a Lite instance. Though, be careful when doing that if you rely on the shared memory. At least one non-lite instance must be running to keep the memory in the grid. Having too few non-lite instances could also result in too much heap of those instances consumed by the shared memory. You can turn an instance to a lite instance with the following asadmin command:

```
asadmin > set-hazelcast-configuration --lite=true
```

Summary of Clustering Options in Payara Server 5

	Domain Data Grid	Deployment Group	Clusters (Deprecated)
Hazelcast-Based	✓	✓	✓
Running member instances visible in Domain Data Grid	✓	✓	✓
Managed only from the DAS	✗	✓	✓
Can be a deployment target	✗	✓	✓
Member instances can be started/stopped together	✗	✓	✓
Supports load balancing and fail-over	✗	✓	✓
Instances can have different configuration	✓ *1	✓	✗
Member instances visible in the list of instances	✓	✓	✗
Instances can connect dynamically (without configuring the cluster)	✓	✗	✗
Compatible with Payara Server 4 cluster admin commands	✗	✗	✓
Can be joined by a Payara Micro instance	✓ *2	✗	✗

1) only if they are in the same domain
2) if Payara Micro started with the same discovery mechanism as Payara Server. By default it uses a different mechanism.

The following entities can join a **Domain Data Grid**:

- Instances that are part of a Deployment Group
- Instances created for a Cluster (Deprecated)
- Instances in a separate domain configured to join the same Data Grid Group
- Payara Micro Instances

The following entities can join a **Deployment Group**:

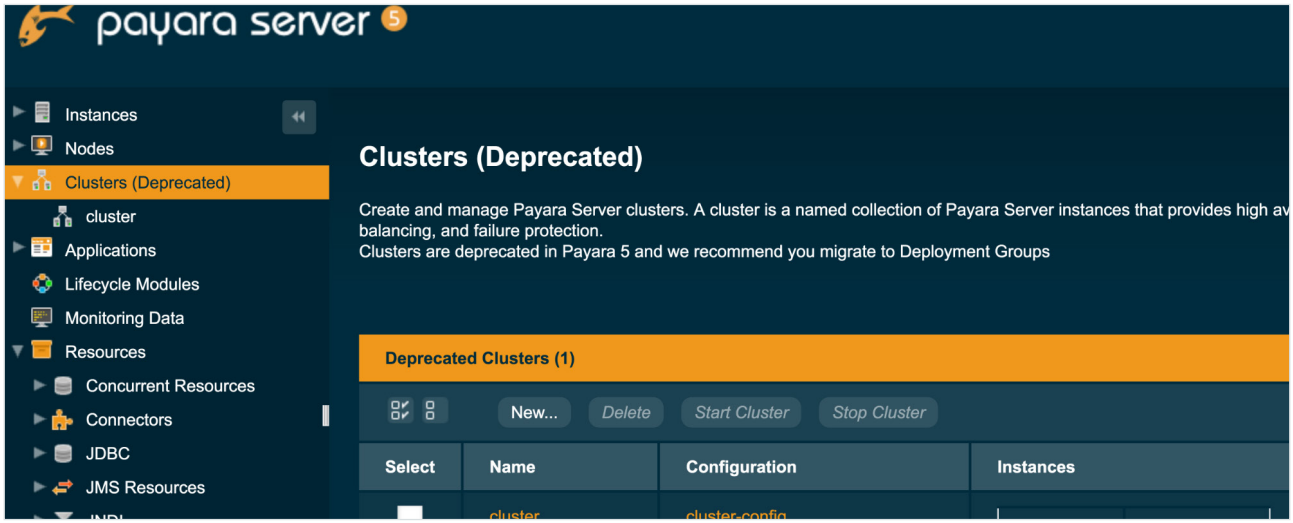
- Instances created directly when creating the Deployment Group
- Instances created separately and added to the Deployment Group

The following entities can join a **Cluster (Deprecated)**:

- Instances created exclusively for the cluster.

Keeping a Standard Payara Server 4 Cluster

As stated previously, Payara Server 5 will understand the configuration of Shoal Clusters migrated from Payara Server 4. If your domain contains a deprecated cluster, you can start that same domain in Payara Server 5 without any changes. The main difference is that Payara Server 5 will use a Hazelcast grid under the covers to provision the cluster instead of the Shoal/GMS technology, otherwise the cluster will function as before with the exception that it's managed in the Admin Console under a page called **Clusters (Deprecated)**:



This is a convenience feature of Payara Server 5 that is used to ease migrations from Payara Server 4. Traditional clusters are now managed in **Clusters (Deprecated)** view on the Admin Console as shown in the image.

If you were established secure communication over SSL/TLS among the instances of your Shoal cluster in Payara Server 4, keep in mind that this feature is not available in Payara Server 5, so you will have to leave the communication channel unsecured. If this is a requirement you must fulfill, it is recommended that you turn in the **Domain Data Grid Encryption** feature (introduced in release *5.201*), which will guarantee that the data that is handled and transferred across instances of the Domain Data Grid is properly encrypted and secured.

To enable this feature, you must generate a private key that will be used by the data grid to encrypt this information:

```
asadmin > generate-encryption-key
```

And then, manually enable the encryption feature:

```
asadmin > set-hazelcast-configuration --encryptdatagrid true
```

More information about the details of how this feature operates can be found in the official Payara Platform documentation.

Migrate to Deployment Groups

Although clusters from Payara Server 4 should work in Payara Server 5, these types of clusters are deprecated, and we recommend you migrate to **Deployment Groups** instead. Look for how to do it in the following sections.

Migrating from a Standard Payara Server 4 Cluster to a Deployment Group

If you decide to migrate to a Deployment Group, you'll get more flexibility in how you manage your cluster. Deployment groups are like clusters but, besides no longer creating instances specific to a cluster, it's possible to create and configure instances individually and later add or remove them from a deployment group. It's also possible to add the same instance to multiple deployment groups.

You can migrate a cluster from Payara Server 4 to a Deployment Group directly during an upgrade to Payara Server 5. Or you can keep the cluster during the upgrade (as described in the previous section) and later migrate a deprecated cluster in Payara Server 5 to a Deployment Group, whichever option best fits your overall migration plan.

If you want to keep the configuration and behaviour of a deployment group as similar as possible to the migrated cluster, follow these steps:

1. Copy any custom cluster configuration
    - If your cluster contains custom configuration, copy it to to the configuration associated with the cluster (e.g. cluster-config)
    - Alternatively, instead, you can note it down to apply it later to a new deployment group
2. Convert all cluster instances to standalone instances
    - While the domain is stopped, manually modify the *domain.xml* file on the DAS and remove the *<clusters>* element completely, including all child elements
    - This will also delete all of the clusters. If you have more clusters, you can only delete the *<cluster>* element in *<clusters>* which corresponds to the migrated cluster

3. Create a deployment group
    - Start the domain
    - Create a deployment group (you may give it the name of the migrated cluster, if the cluster no longer exists)
    - Add the instances, which belonged to the previous cluster, into the new deployment group
    - Create any custom resources on the deployment group if needed (if you didn't copy them to the cluster configuration earlier)

You can now do the same actions on the new deployment group like you could do on the previous cluster. For example, the following actions are equivalent:

Action	Deployment group		Cluster	
	Admin Console	asadmin command	Admin Console	asadmin command
Start all instances	Start Deployment Group	`start-deployment-group`	Start Cluster	`start-cluster`
Stop all instances	Stop Deployment Group	`stop-deployment-group`	Stop Cluster	`stop-cluster`

Create an instance	New instance in the group.
	
	And existing instance to the group
		`create-instance`
		
		`add-instance-to-deployment-group`
			New instance in the cluster	`create-instance --cluster`

Some configuration that's available for clusters is also available for deployment groups. This configuration is applied on top of the configuration of each server instance in the group, such as deployed applications, resources and properties. All other configuration settings that are missing for a deployment group can be applied to the configuration of individual instances by editing their configuration (e.g. a configuration named cluster-config) or by other means:

- Batch configuration is available in the Batch page of each individual configuration (in the sidebar in Admin Console)
- JMS Physical destinations can no longer be configured from within Payara Server. Instead, you can use the *imqadmin* or *imqcmd* tools in the *mq/bin* directory and connect to an MQ server used by the deployment group directly. To add a JMS resource to the whole deployment group, add the deployment group to the resource's targets

Migrating from a Hazelcast Cluster of Standalone Instances to the Domain Data Grid

In Payara Server 4, the recommended way to create a cluster backed by the Hazelcast mechanism is to create multiple standalone instances that reference the same clustering configuration so that they connect between each other. If this is your case, this section will explain how to migrate it to the Domain Data Grid with a similar setup in Payara Server 5. Once this is done, you have the option to group your instances into deployment groups to manage them as a single cluster, which isn't possible in Payara Server 4.

No configuration changes are needed to migrate standalone instances in Payara Server 4. If this is the case for your Payara Server 4 domain, you can just use it on Payara Server 5 without any configuration changes. This will result in having the same instances in Domain Data Grid and all would work as in Payara Server 4 with some changes described below:

- Instead of using multicast, the Hazelcast discovery mechanism is changed to the *domain* discovery mode introduced in Payara Server 5. On this mode, the Domain Admin Server (DAS) won't connect to any instance, instead, instances will "talk" to the DAS to join the Data Grid. This means that the DAS should be started before any instance, otherwise it can take up to 5 minutes until instances started before the DAS can connect to the data grid. This time can be decreased by modifying the Hazelcast merge delay system properties. The discovery mechanism can also be changed back to using *multicast* discovery in the Admin Console configuration for the Domain Data Grid configuration or by executing the asadmin command `set-hazelcast-configuration --clustermode multicast`
- The DAS listens on a different Hazelcast port than server instances. By default, it listens on port **4900** while other instances listen on ports starting with **5900**. In Payara Server 4, both DAS and instances use ports starting with **5900** by default.
- Payara Micro instances won't automatically connect to the Payara Server 5 Domain Data Grid with default configuration.

Keeping a Cluster of Payara Micro Instances

If you cluster together multiple Payara Micro instances, no changes in configuration are needed for running it with Payara Micro 5. **The discovery mechanism in Payara Micro 5 is still multicast**, therefore instances will discover and form a cluster themselves as before. However, there are some caveats:

- The default starting port changed from **5900** to **6900**. To change it back, use the `--startport` command line argument when starting each instance
- The default multicast address and port for multicast discovery are different in Payara Server and Payara Micro. To modify them in Payara Micro, use the `--mcport` and `--mcaddress` command line arguments when starting each instance.
  - in Payara Server, the default *mcaddress* (multicast group) is 224.2.2.3 and mcport (multicast port) is 54327. Run Payara Micro with these values if you didn't modify them on Payara Server
- It's now easier to configure a different discovery mechanism than multicast with the `--clustermode` command line argument. On Payara Server 4 you must configure a customized *hazelcast.xml* configuration file and manually configure each discovery mode

Keeping a Hybrid Cluster of Payara Server and Payara Micro Instances

Payara Micro 4 instances can automatically join a cluster of Payara Server 4 instances and use the DAS of the server to monitor the list of instances conforming the cluster. This is usually known as a *Payara Hybrid cluster*. Payara Micro 5 instances on the other hand won't automatically cluster with a Payara Server Domain Data Grid, due to the changes mentioned in the previous section. If you have a hybrid cluster that you want to migrate to the Payara Platform, there are two options to consider:

- Configure the Payara Server Domain Data Grid to use the `multicast` discovery mode so that you get the same clustering behavior as in Payara Server 4
- Configure the Payara Micro instances to use the `domain` discovery mode so that they connect to the Domain Data Grid

> ***You need to upgrade both Payara Server and Payara Micro instances to the same version at once. If you don't, you might encounter unexpected side effects and loss of data within the cluster.***

The preferred way in most cases is to use the first option: The new `domain` discovery mode; it's more powerful, works in any network topology and even works well in cloud environments. This discovery mode only requires either the IP address or host name of the DAS. To do this, start your Payara Server 5 domain, wait until the Domain Data Grid is fully started by checking the DAS' log and lookin for the following (similar) entry:

```
INFO:   Data Grid Status
Payara Data Grid State: DG Version: 35 DG Name: development DG Size: 1
Instances: {
  DataGrid: develpoment Instance Group: MicroShoal Name: server Lite: false
This: true UUID: b029cd04-5ffc-4082-b57c-2865e104d82c
Address: /192.168.1.148:4900
}
```

And then start your Payara Micro 5 instances like this:

```
java -jar payara-micro-5.jar --clustermode domain:<das-hostname>:<cluster-port>
```

And that's it. The Payara Micro instances will join the Data Grid and will show in the status of the Domain Data Grid either on the DAS' log or in the list shown in the admin console.

> ***It's not required that the DAS is running when starting new Payara Micro instances. They won't join a cluster and will wait until the DAS is accessible to join the cluster. However, there may be up to a 2-minute delay before the DAS is started and each instance finds out about it so it's better to run DAS while Payara Micro instances are started.***

If you'd rather keep the multicast discovery as with Payara Server 4, you need to change the Domain Data Grid Discovery mode to `multicast` on Payara Server 5. You also need to run Payara Micro with the `--mcport` and `--mcaddress` command line arguments to specify the same multicast address and port used by Payara Server because they are different by default. Keep in mind that this will affect the way other Payara Server 5 instances will use to join the Domain Data Grid as well. Use this option only if you have careful control of all your instances, be they Payara Server or Payara Micro instances.

Mapping Between JSON and Java Objects

Description of the Changes in JSON Mapping

You can run Java EE 8 applications on Payara Server 5. One of the main benefits of this new version of the Java EE standard is that it includes the JSON-B (JSON Binding) API. This API is used to define a serialization of POJOs into JSON payloads and vice-versa.The Jackson library is a commonly used third-party alternative which served as inspiration for this new API. One of the main advantages of JSON-B in Payara Server 5 is that it is integrated out of the box with the JAX-RS container. It is used for automatic serialization and de-serialization of POJOs that are part of the payload managed in both JAX-RS REST service requests and responses. All of this is defined by standard JSON-B mapping and doesn't rely on a non-standard mapping provided by custom extensions.

Payara Server 4 also provides automatic mapping between Java objects and JSON within the JAX-RS container, but compared to Payara Server 5, this mapping is derived from JAX-B (Java XML Binding) API, which is designed for mapping between Java objects and XML and isn't convenient for the JSON format. Furthermore, while this mapping is standardized for XML payloads, it's not generally supported on other application servers for JSON payloads. In Payara Server 4, the default implementation of a JSON serialization provider for JAX-RS (Jersey) is an EclipseLink library called MOXy provided by the EclipseLink component. Jackson also provides a JAX-RS mapper which works very well with Payara Server 4 and is often used as an alternative to Moxy.

When migrating applications from Payara Server 4 to 5, if your applications declare JAX-RS components that rely on the automatic serialization and marshaling mechanism provided by JAX-B annotations, keep in mind **that these annotations will be ignored** on Payara Server 5 by default! This is due to the switch from JAX-B to JSON-B as the default provider for JAX-RS JSON payloads.

Migrating from JAX-B Mappings to JSON-B Mappings

Because JSON-B is a standard way of mapping Java objects to JSON payloads it's best option when building new applications with Payara Server 5. If you are migrating an application from Payara Server 4, it might not be convenient to refactor all your JAX-B mappings to use it but we still recommend to evaluate this option. Refactoring to using JSON-B would give you the advantage of using a well -integrated and better supported API for JSON mappings and confidence that your application uses standard and predictable API that will not break after upgrades to newer versions of Payara Server or even in case of migration to any other server.

If you want to maintain the same configuration for the JSON serialization and marshalling of your entities in your applications with JSON-B as you now have with JAX-B, you will have to refactor your code to use the corresponding JSON-B annotations.

Applications that use the JAX-B API for configuring JSON serialization in Payara Server 4 (using the default MOXy provider) will have annotated classes like this one:

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement(name = "payload")
public class MyPayload {

    @XmlElement(name = "payloadID", required = true)
    private String id;

    @XmlAttribute(name = "editable")
    private Boolean editable;

    ...
}
```

The same entity configured using JSON-B in Payara Server 5 would look like this:

```
public class MyPayload {

    @JsonbProperty(value="payloadID", nillable=false)
    private String id;

    private Boolean editable;

    ...
}
```

You will notice that the same entity configured using JSON-B annotations is easier to understand and has less declarations. For getting started with the JSON-B API, you can follow the official Getting started with JSON Binding guide.

It's also possible to keep the JAX-B annotations together with the new JSON-B annotations. This is recommended if:

- your application maps the same entity to both XML and JSON
- you want to compare how the new JSON-B mapping performs compared to the JAX-B mapping
- you want to keep the possibility to easily revert to using the JAX-B mapping with Payara Server 5 in the future

Keep Using JAX-B Mapping for JSON in Payara Server 5

It's understandable that in some cases, refactoring definitions for mapping between Java classes and JSON payloads can require a lot of effort. In this case, it's possible to configure your application to still use the JAX-B annotation configuration as it was on Payara Server 4. To do this, you'll have to add the following Servlet context parameter to your web.xml deployment descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.
jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1" metadata-complete="false">
    <servlet>
        <servlet-name>com.mycompany.MyApplication</servlet-name>
        <init-param>
            <param-value>MoxyJsonFeature</param-value>
        </init-param>
    </servlet>
    ...
</web-app>
```

With this configuration, JAX-RS services in your application will support the same mapping with JAX-B annotations as supported in Payara Server 4. In addition, JSON-B annotations in the same application if there are any, would be ignored. Keep in mind that in the future the JAX-B annotation support might be dropped entirely in the future, so it's best that, at some point, you plan to refactor your applications to use JSON-B annotations instead.

Keep Using Jackson 2 Library for JSON Mappings in Payara Server 5

If you use Jackson 2 library for mapping Java classes to JSON payloads in JAX-RS endpoints, you can keep using it in Payara Server 5. In the long run, we recommend to migrate to using JSON-B API because it would give you the advantage of using a well-integrated and supported API for JSON mappings, and confidence that your application uses a standard and predictable API that will not break after upgrades to newer versions of Payara Server (or even in case of migration to another server running Java EE 8 applications). As a bonus, your application would be thinner after you drop the Jackson dependency.

To continue using Jackson with Payara Server 5, you'll have to add the following Servlet context parameter to your *web.xml* deployment descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.
jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1" metadata-complete="false">
    <servlet>
        <servlet-name>com.mycompany.MyApplication</servlet-name>
        <init-param>
           <param-name>jersey.config.jsonFeature</param-name>
           <param-value>JacksonFeature</param-value>
        </init-param>
     </servlet>
    ...
</web-app>
```

In the case you define the JAX-RS application programmatically, you can add the JacksonFeature to the Classes list of your Application.

```
@ApplicationPath("/api")
public class JaxRsActivator extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> classes = new java.util.HashSet<>();
        classes.add(com.fasterxml.jackson.core.util.JacksonFeature.class);
        return classes;
    }
}
```

You also needed to have all required Jackson 2 dependencies in your application. This is also required by Payara Server 4 so the Jackson dependencies should already be in your application and you don't have to take any further action.

With this configuration, JAX-RS services in your application will support the same mapping with the Jackson 2 annotations as supported in Payara Server 4. In addition, JSON-B annotations in the same

application if there are any, would be ignored. Note that this configuration is very similar to enabling the JAX-B mappings described in the previous section.

Built-in Databases

Payara Server 5 includes the **H2 database**, which isn't present in Payara Server 4. The H2 database is used for the default JDBC data source for applications instead of the Derby DB used in Payara Server 4. A data source for Derby DB is still present in Payara Server 5 but usage of Derby DB is **deprecated and not supported.**

Description of the Changes in Built-in Databases

Some of the internal features of Payara Server, either features exposed as part of the standard set of APIs or features specific to Payara Server, require the use of a data store to persist data after the server shuts down. Payara Server 4 comes bundled with Derby database (also known as JavaDB) that is used to make these features work correctly, and this database is exposed as a set of two JDBC resources:

- A connection pool called `__TimerPool` which connects to an embedded Derby database. Since this is an embedded database, the server will start this database inside the same JVM process used for the DAS. This pool is exposed as `jdbc/__TimerPool` JDBC datasource but is not to be used by applications.
- A connection pool called `DerbyPool`, which connects to a standalone Derby database. It's associated with the JDBC data source identified by `jdbc/__default`. This data source is used as the default data source for applications.

However, Derby DB is currently considered an outdated product with several production-aware issues (like inconsistent concurrent updates and unexpected row-locking). These issues motivated us to gradually replace this database with a more robust solution in Payara Server 5, which is H2 database. The following is a list of the changes introduced in Payara Server 5:

- There is a new JDBC connection pool called `H2Pool` which is used to configure database connections to an embedded H2 database.
- The `jdbc/__default` JDBC data source is now linked to this new connection pool instead of the `DerbyPool` connection pool
- The `__TimerPool` connection pool is configured to connect to an embedded H2 database as well but it uses an XA Data source resource configuration to allow multiple instances to

connect to it concurrently. This connection pool is intended to be used by the EJB Timer Service separately from the default connection pool.
- The DerbyPool connection pool no longer exists.
- There is a new connection pool named SamplePool that is configured to connect to a local Derby server. This connection pool can be used to quickly connect to a local unsecured Derby database for development purposes only.
- There is a new JDBC data source named jdbc/sample that uses the previous connection pool for the local Derby database.
- The asadmin commands start-database and stop-database will control the life cycle of the local H2 database instance instead of the old Derby database

> ***Production Environments***
>
> ***Neither Derby DB nor H2 DB are recommended for production usage. H2 is included within Payara Server to simplify application development. For production environments, we recommend using a separate production-ready relational database configured as a JDBC resource and its corresponding JDBC connection pool.***

H2 Database

H2 DB is installed in the directory `${PAYARA_INSTALL_DIR}/h2db`.

To start the standalone H2 database, use the following asadmin command:

```
asadmin> start-database
```

To stop the H2 database, use the following asadmin command:

```
asadmin> stop-database
```

Derby Database

There is no embedded Derby installation in Payara Server 5.

Keeping the Data Source Configuration from Payara Server 4

As stated previously, usage of Derby DB is **not supported**. If you are upgrading to Payara Server 5 with your domain configuration copied directly from a Payara Server 4 domain, the data source and connection pool configuration will not work correctly due to the embedded Derby installation missing

from the server files, so it is recommended that you migrate the old data source and connection pool configuration settings to the ones that rely on H2 instead. Read the following section to find out how.

Migrating to the New Data Source Configuration in Payara Server 5

If your domain relies on the default data source, you need to make sure that your applications will work with the H2 DB instead of the Derby DB. To do this, follow these steps:

1 – Before the domain is started, proceed to manually modify the domain.xml configuration file and edit the default connection pools. Locate the resources tag element and identify the default connection pools:

```
<resources>
    <!-- … -->
    <jdbc-connection-pool datasource-classname="org.apache.derby.jdbc.
EmbeddedXADataSource" name="__TimerPool" res-type="javax.sql.XADataSource">
      <property name="databaseName" value="${com.sun.aas.instanceRoot}/lib/
databases/ejbtimer"></property>
      <property name="connectionAttributes" value=";create=true"></property>
    </jdbc-connection-pool>
    <jdbc-connection-pool is-isolation-level-guaranteed="false" datasource-
classname="org.apache.derby.jdbc.ClientDataSource" name="DerbyPool" res-
type="javax.sql.DataSource">
      <property name="PortNumber" value="1527"></property>
      <property name="Password" value="APP"></property>
      <property name="User" value="APP"></property>
      <property name="serverName" value="localhost"></property>
      <property name="DatabaseName" value="sun-appserv-samples"></property>
      <property name="connectionAttributes" value=";create=true"></property>
    </jdbc-connection-pool>
    <!-- … -->
</resources>
```

Now, replace these definitions with the default connection pool settings used for the H2 databases:

```
<resources>
    <!-- … -->
    <jdbc-connection-pool datasource-classname="org.h2.jdbcx.JdbcDataSource"
name="__TimerPool" res-type="javax.sql.XADataSource">
      <property name="URL" value="jdbc:h2:${com.sun.aas.instanceRoot}/lib/
databases/ejbtimer;AUTO_SERVER=TRUE"></property>
```

```
    </jdbc-connection-pool>
    <jdbc-connection-pool is-isolation-level-guaranteed="false" datasource-
classname="org.h2.jdbcx.JdbcDataSource" name="H2Pool" res-type="javax.sql.
DataSource">
      <property name="URL" value="jdbc:h2:${com.sun.aas.instanceRoot}/lib/
databases/embedded_default;AUTO_SERVER=TRUE"></property>
    </jdbc-connection-pool>
    <!-- … -->
</resources>
```

2 – Locate the default JDBC resource definition for the **default datasource only** in the same tag element:

```
<resources>
    <!-- … -->
    <jdbc-resource pool-name="DerbyPool" object-type="system-all" jndi-
name="jdbc/__default"></jdbc-resource>
    <!-- … -->
</resources>
```

Replace its definition with the Payara Server 5 equivalent:

```
<resources>
    <!-- … -->
    <jdbc-resource pool-name="H2Pool" object-type="system-all" jndi-
name="jdbc/__default"></jdbc-resource>
    <!-- … -->
</resources>
```

Keep in mind that when changing the internal database from Derby to H2 will make the data that is stored in the old database inaccessible by the new state of the server. In most cases this won't be a problem since currently Payara Server uses this internal database to store the information of persistent timers and the historic information of executed batch jobs. Persistent timer information can be skipped without issues in a controlled migration (the server will create new data if the database is empty), so the only relevant set of data that you might be interested to keep would be historic batch jobs data.

If you are interested in keeping this data, our recommendation is that you export this data by creating the relevant SQL data manipulation scripts that inserts the data in the H2 database. Both Derby and H2 have a similar SQL syntax, so this shouldn't take that much effort.

HTTP/2 Protocol Support

Payara Server 5 introduces support for the HTTP/2 protocol as part of the new Servlet 4.0 API. This protocol is enabled by default on the default HTTPS network listeners included within the server's configuration.

Changes Related to HTTP/2 Protocol

Support for HTTP/2 protocol is enabled on secure HTTP network listeners **by default** in Payara Server 5. This version of HTTP protocol brings a lot of performance improvements like:

- Request and response multiplexing to reduce the number of required connections
- Header compression to reduce the amount of data
- Server Push to send multiple related files faster
- Binary encoding of commands to improve security

Additionally, the protocol requires encryption with an improved version of Transport Layer Security (TLSv1.2 at a minimum). That's why it can only be enabled on secured HTTP network listeners in Payara Server.

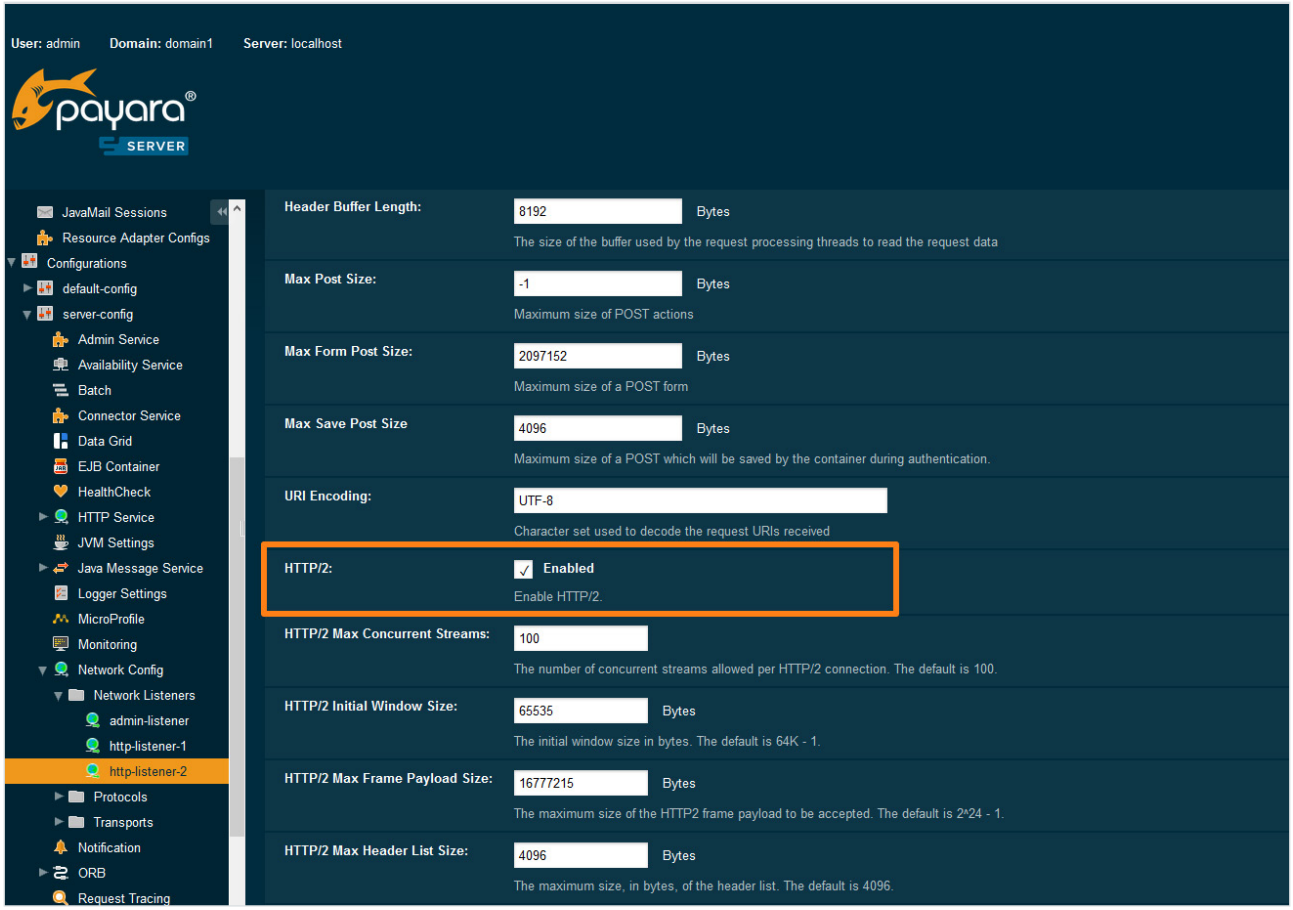
While some web frameworks used for client applications and web browsers can leverage HTTP/2 features to improve overall network performance, HTTP/2 support is not guaranteed to be stable enough in all cases, which could cause issues for your applications. If you are upgrading an existing application from Payara Server 4 to Payara Server 5, we recommend disabling HTTP/2 support on all HTTP listeners first to avoid encountering unwanted errors. After your application runs successfully on Payara Server 5, you can test it with HTTP/2 enabled to verify if it doesn't introduce any issues.

By design, HTTP/2 doesn't support authentication using client certificates. In HTTP/2, a client can have multiple outstanding requests. Without some sort of correlation information, a client is unable to identify which request caused the server to request a certificate. If you need to use client certificates for authentication, then you should disable HTTP/2 and keep it disabled.

Keeping HTTP 1.1 Protocol for All Listeners

The safest way to upgrade from Payara Server 4 to Payara Server 5 is to keep the same configuration for all HTTP listeners using the HTTP 1.1 protocol untouched and ensure that HTTP/2 is disabled completely.

If you are using the Admin Console, you can disable the HTTP/2 protocol for network listeners in the *Network Config* → Protocols option. After choosing the listener, go to the HTTP tab and un-select the HTTP/2 option:



If you prefer using the command line, you can disable the HTTP/2 protocol on a network listener by executing the following asadmin command:

```
asadmin> set configs.config.server-config.network-config.protocols.
protocol.<listener-name>.http.http2-enabled=false
```

Known Issues After Migrating

There are a few known issues at the moment that can arise in your environment after executing a successful migration to Payara Server 5, which are listed in the following table along with their causes and recommended workarounds:

Issue	Cause	Workaround
Admin Console doesn't show monitoring data for instances apart from the DAS	Internal changes in the Jersey components used by the admin console.	To access the monitoring data of all relevant instances it's best to use the REST monitoring API offered by the DAS and query the relevant data separately in the browser. This is a known bug and will be fixed in a future release of Payara Server 5.
PrimeFaces applications encounter unexpected errors	The Server Push feature of the HTTP/2 protocol is known to interfere with PrimeFaces' pushing mechanisms	Disable the Server Push features in all relevant HTTP network listeners. There is no clear solution at the moment in order to make both features work in co-existence, so if there are other applications that require the use of HTTP/2 Server Push, it's best to define a customized virtual-server where the application should be deployed within a separate network listener.

Client Certificate Authentication does not work when used with HTTP/2	By design, the protocol does not support the `CLIENT-CERT` authentication method	Disable the HTTP/2 protocol in all relevant network listeners. If HTTP/2 usage is a priority, switch out to a better suited authentication method.

Conclusion

After following the instructions detailed in this guide you should be able to run your old Payara Server 4 domains in Payara Server 5. Keep in mind that these instructions showcase the necessary steps to have a working domain in Payara Server 5, so additional features that can be used to increase the productivity of your applications should be considered when further developing them. We recommend that you browse through the official product documentation to have a better understanding of these features.

Lastly, this guide covers the most common scenarios and challenges that can be encountered when implementing a server migration; so in the case you stumble around a different issue than the ones documented here, please make sure to raise a support ticket if you have a Payara Enterprise subscription (by using the Customer Hub support portal or writing an email to support@payara.fish) and include all of the relevant information about the problem you are encountering. We'll work in helping you overcoming these issues and complete a successful migration!

Where to Get More Migration Help

Hands-On Assistance for Payara Enterprise Customers

For additional help with the migration from Payara Server 4 to Payara Server 5 as a Payara Enterprise customer, download our Payara Accelerator Upgrade Guide, and learn about our add-on consultancy solution. Download our Payara Accelerator Upgrade Guide to learn more about our consultancy solution.

Get Payara Platform Enterprise

If you're not yet a Payara Enterprise customer, obtaining a Payara Enterprise subscription will give you access to Payara Platform experts to accelerate your project delivery while reducing risks and costs and helping you deliver your project on time and within budget.

You'll have unlimited tickets to get all of your questions answered, access to a private customer knowledge base and exclusive access to crucial fixes and patches.

Learn more about Payara Platform Enterprise.

sales@payara.fish

+44 207 754 0481

www.payara.fish