GlassFish to Payara Server 5 Migration Guide

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

Migration Guide

Contents

Introduction

Since Sun Microsystems first developed GlassFish as the reference implementation (RI) of Java EE, it has been used by both developers in its open source form, and by Sun - then, later, by Oracle - in production, fully supported. In a competitive market, GlassFish has fared well and become a popular choice, with the vast majority using the GlassFish Server Open Source Edition. In late 2013, Oracle announced the end of commercial support for GlassFish. By the time of publication of this guide, both Premier and Extended support options for Oracle GlassFish have expired and as such, unless the user has a sustaining support contract; no official support for the product is available.

Obviously, Oracle's decision has presented a problem: both for users of the open source edition, and licensees of Eclipse GlassFish. There have only been three Java EE 7 certified releases of GlassFish Open Source Edition since 2013, and concerns are arising for the future quality of GlassFish, due to the lack of commercial support users raising bugs and funding the fixes for the community, as well as other paying users.

One year after the original announcement of the end of support from Oracle, Steve Millidge - a Java EE expert and the founder of Payara Services Ltd - announced the arrival of Payara Server and, with it, a reintroduction of a 24/7 vendor support option for businesses who had previously chosen GlassFish as a platform. Payara Server proved to be the logical 'drop-in replacement' for GlassFish Server Open Source Edition, helping companies migrate quickly and easily onto a very similar platform with added reassurance of regular releases, bug fixes, patches and enhancements.

As Payara Server has developed, its popularity has grown very quickly among the community due to an open and responsive development team, a regular quarterly release cycle and introduction of some crucial production enhancements which were never available in GlassFish.

Migrating from GlassFish to Payara Server can be a simple and straightforward process, made even simpler with the help of this Guide. With no need for code rewrites or application re-architecting, Payara Server is a credible solution on which to build your Java middleware platform. On the next pages, you will find an overview of the things you will need to consider in your GlassFish to Payara Server migration project; as well as details of Payara Server's tools and features which will make your life (and your application server management) much easier!

Why Choose Payara Server Enterprise?

The first decision to make when facing a migration is to consider what platform to migrate to. Staying with a newer version of the same server is the most pain-free path to take. In the case of GlassFish migration, Payara Server is the natural successor, **since the source code was originally derived from GlassFish and adapted for modern software architectures** - it is a technology that you and your developers are already familiar with.

Enterprise quality production and development enhancements (detailed further in this guide), **easy** installation, **powerful** administration interface, **simple to use** tools and features, monthly release cycle, backed by emergency hot fixes to continually improve application **security** - these are just a few examples of the many great features of Payara Server that make it an ideal solution for current GlassFish users.

On top of that, Payara Server is entirely open source and - unlike the commercial edition of GlassFish - there are no components withheld from the community. From the beginning, Payara strives to be open and transparent. All the source code and commits are publicly available, and community users are welcome to raise issues against the server (see GitHub) to help increase quality. Since the source code is evolutionary over what is available in GlassFish 4.x, the resulting container is a more natural choice for an upgrade than GlassFish 4.x or 5.x itself. Payara Server fixes a lot of defects present in GlassFish and offers performance improvements along with many new features that offer a smooth migration experience. Payara Server 4.x is a drop-in replacement of GlassFish 4.x and migrating to it is very similar to migrating to GlassFish 4. Payara Server 5 introduces changes in some areas, most notably to improve clustering features, so migrating to it might require a bit more effort than migrating to GlassFish 5.x. But, again, the migration to Payara Server 5 is often with much better results because of many improvements, bug fixes and new features that add better support for modern technologies like Docker, cloud deployments and microservices.

Application servers are complex pieces of software and have many areas where edge-case bugs can hide. Payara Server Community offers innovative, monthly releases with new features and enhancements to try out, while Payara Server Enterprise offers monthly builds to maintain the security and stability of our customers' environments. The frequency of updates provides strong reassurance that any immediate issues are much more likely to be fixed rapidly. So, unlike the users of GlassFish, Payara Server users do not have to wait 12 (or more!) months for a new release with no guarantee that a fix will be included.

Providing a logical migration path from GlassFish Server Open Source Edition, Payara Server is the platform of choice for your Jakarta EE (Java EE) applications. Our vision is to optimise Payara Server to make it the best application server for production Jakarta EE applications.

Migration Options from GlassFish 3 to Payara Server 5

When migrating from the commercial edition of GlassFish 3 to Payara Server 5, there are these areas to consider:

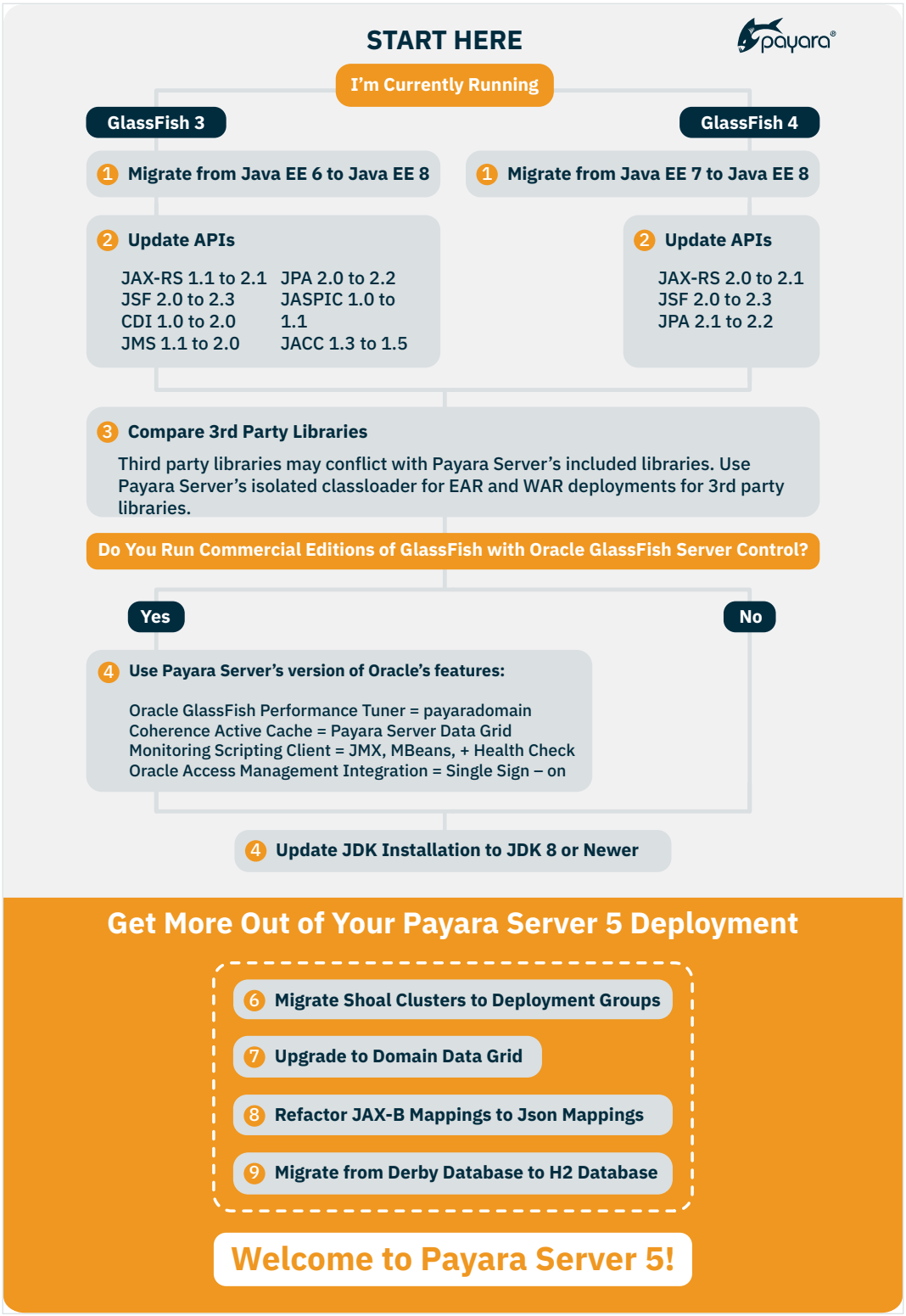
- How to deal with Oracle commercial features, which aren't available in Payara Server
- How your application might be affected by API changes in Payara Server
- How to migrate your GlassFish configuration to Payara Server
- How to get the most out of the new features and improvements in Payara Serve

In some cases, the migration might be simpler to do in 2 steps:

- Migrate from GlassFish 3 to Payara Server 4
- Migrate from Payara Server 4 to Payara Server 5

When migrating from the open source GlassFish edition, you don't need to consider migrating from Oracle's commercial features, but all other areas are still valid.

Migrating from GlassFish 3.x to Payara Server 5.x



As far as the application server itself is concerned, there were very few major changes between GlassFish 3 (Java EE 6) and 4 (Java EE 7). This also means there are very few major changes between GlassFish 3 and Payara Server 4. There were a lot more changes in Payara Server 5, mainly in the following areas:

- JDK 8 is required
- Updates between Java EE 6 and Java/Jakarta EE 8
- Supports MicroProfile and JCache APIs
- Clustering and High Availability changes
- A new default embedded database
- HTTP/2 protocol by default on secure listener

Payara Server 5 includes several updated APIs and regularly incorporates updated minor versions of any external projects like, for example, Weld for CDI or Mojarra for JSF. There have been a couple of updated APIs in Java EE 8, which were released through GlassFish, however not all of these are significant changes.

APIs with significant updates between Java/Jakarta EE 6 and Java EE 8 include:

- Servlet, from 3.0 to 4.0, with HTTP/2 and Server Push support
- JAX-RS, from 1.1 to 2.1
- JSF, from 2.0 to 2.3
- CDI, from 1.0 to 2.0, with asynchronous events
- EL (Expression Language), from 2.2 to 3.0
- Bean Validation, from 1.0 to 2.0
- JMS, from 1.1 to 2.0
- JPA, from 2.0 to 2.2
- JASPIC, from 1.0 to 1.1
- JACC, from 1.3 to 1.

As always, backwards compatibility is of high importance to Java EE so, for example, a JMS 1.1 MDB should still be able to interact with a modern, JMS 2.0 enabled broker. Any code still using older APIs would be supported fully by the Payara support team in the case of any bugs found in Payara Server.

There are some brand-new APIs, as well as updated existing APIs, released in Java EE 7 in GlassFish which Payara Server 4 has inherited. These include:

- Concurrency
- JBatch
- Websocket
- JSON-P
- JSON-B
- Java/Jakarta EE Security

In addition to APIs which are part of the Java EE 8 specification, Payara Server 5 also implements the following APIs:

JCache which, though the specification has yet to be incorporated into Java EE, is a standard implemented by many vendors already. Payara Server provides JCache support by embedding Hazelcast and can therefore make use of dynamic clustering to store session data.

MicroProfile is a set of APIs that integrates well with Java EE APIs. It specifies how monitoring and other information about applications is exposed by Payara Server and provides APIs for implementing microservices, reactive, and other patterns not yet covered by Java EE. Payara Server 5 implements MicroProfile 3.x APIs.

The new APIs will be useful to developers but are of no immediate concern for any migration plan. You may want to keep them in mind for future application development planning.

Migrating from GlassFish 4.x to Payara Server 5.x

In most cases, any application which runs on GlassFish 4 will run on Payara Server 4. Therefore, you can follow a dedicated guide about upgrading from Payara Server 4 to Payara Server 5. Or you may go back to the instructions for upgrading from GlassFish 3.x, continue following this guide and skip parts which are not relevant for you. Because GlassFish 4 doesn't bring any new significant features on top of GlassFish 3, steps for migrating of all APIs and features of GlassFish 4 to Payara Server 5 would be the same as for migrating them from GlassFish 3. The only potential additional hiccup may be where the application to be migrated happened to exploit some incorrect or invalid behaviour in GlassFish 4, which has subsequently been corrected or fixed in Payara Server 5.

Payara Enterprise customers can request support to either assist in changing the application concerned or providing some other workaround. The main things to consider when migrating from GlassFish 4.x to Payara Server 4 is not what changes might be mandated - since the number is likely to be zero - the key consideration would rather be what changes are possible, due to the extra features added, such as JCache.

Migrating from GlassFish 3.x to Payara Server 5.x in 2 Steps

Because, compared to any version of GlassFish, Payara Server 5 introduces a lot more changes than Payara Server 4, in some cases it might be easier to migrate first to Payara Server 4 and then to Payara Server 5. This guide focuses on migrating directly from GlassFish 3 to Payara Server 5. If you want to migrate in 2 steps, we recommend you to follow the separate guides about migrating from GlassFish to Payara Server 4 and then from Payara Server 4 to Payara Server 5.

Note that Payara Server 4 is not maintained for the community anymore, and the last community version 4.1.2.181 was released in February 2018. Payara Enterprise customers have access to the latest features and fixes in Payara Server 4 until the year 2025, along with assistance from the support team during your migration, so keep this in mind when choosing which version you will use for the long term.

Migrating from GlassFish Server Control

Oracle GlassFish Server Control is a suite of proprietary features available with the commercial edition of GlassFish 3 that improves performance of the server, enables fine-grained monitoring and enables more secure and highly available production deployments. GlassFish Server Control is composed of the following six features:

- Load Balancer Configurator Plugin
- Domain Administration Server Backup and Recovery
- Coherence Active Cache
- Monitoring Scripting Client
- Oracle Access Management Integration
- Performance Tuner

These features are not available for Payara Server. If you use them intensively with GlassFish 3, there are tools and techniques available that can reproduce the functionality of these features within Payara Server and achieve similar results. They will be described in detail further in this guide.

Main Advantages of Payara Server 5

Payara Server 5 was introduced at the beginning of 2018 as the next major version of Payara Server. Payara Server 4 was derived from 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, deviates 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 deployments
- 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 feature

The purpose of this guide is to help you prepare and understand the main challenges you may face when migrating from GlassFish 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 case. Payara 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 don't yet have a Payara Enterprise subscription, we invite you to take a look at becoming a customer and choosing from our included Migration & Project Support option, 24x7 support, or 10x5 support option.

Migration Process to Payara Server 5

Preparation

Payara Server 5 is **compatible with both JDK 8 and JDK 11**. If your GlassFish domains are currently running on JDK 7 or a lower version, you will have to update your JDK installation to JDK 8 before starting the migration. We encourage you to use Zulu JDK, since Payara Enterprise customers have commercial support for its Enterprise solution as well.

You also must keep in mind that Payara Server 5 supports Java/Jakarta EE 8 applications. When migrating your GlassFish 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.

Migrating a Domain from GlassFish using Backup and Restore

One of the recommended strategies that you can use to migrate your working GlassFish domain to Payara Server 5 is to 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 suspend or stop the GlassFish domain if it's running. Open source GlassFish versions do not support suspending the domain. For them, 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 GlassFish 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:

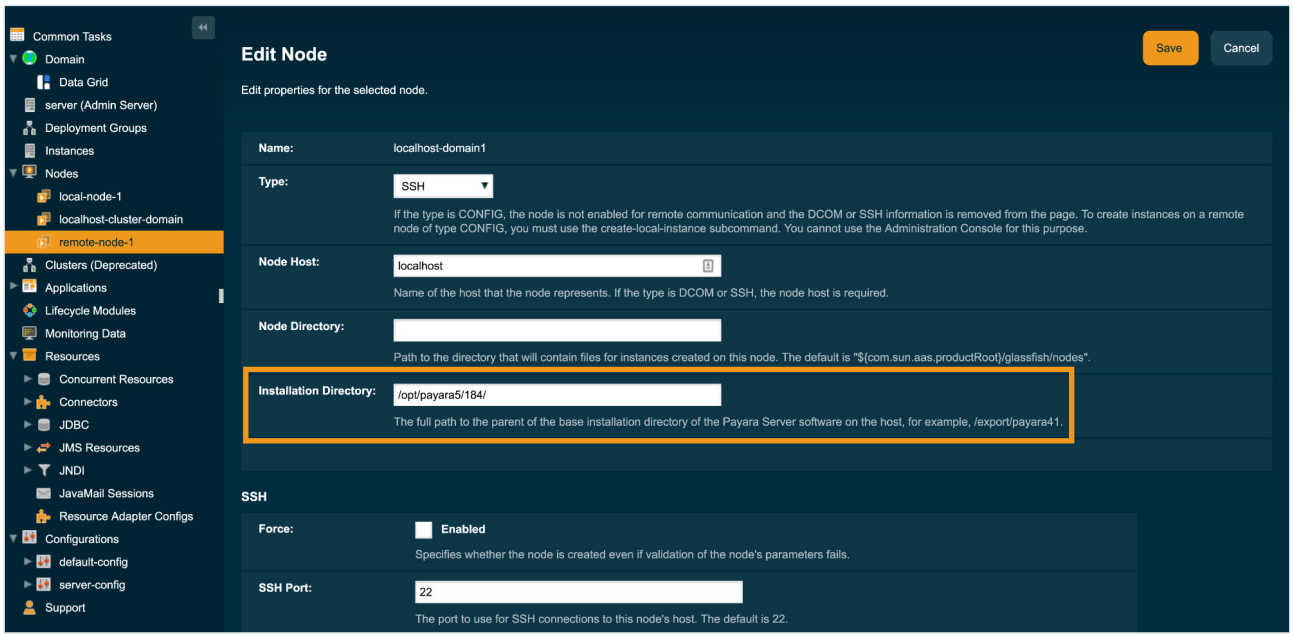
```
Restored the domain (<domain-name>) to /opt/payara5/201/glassfish/
domains/<domain-name>
Description                : <domain-name> backup created on <yyyy_mm_dd> by
user <username>
GlassFish Version          : Payara Server 5.201 #badassfish (build 512)
Backup User                : <username>
Backup 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/payara5/201/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 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 GlassFish binaries in that case, which means that you must forego your working GlassFish domain. This could be a problem if you want to revert your Payara Server 5 installation and go back to GlassFish, so in order to prevent that it is 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 in GlassFish

The clustering mechanism supported by GlassFish is based on the (already deprecated) Shoal project. 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

Clustering and High Availability Improvements in Payara Server 5

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 that its performance is degraded during high load peaks. Payara Server comes equipped with the concept of a **Domain Data Grid** which has the following responsibilities:

- Share the data across all 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 D**eployment 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 provides many more flexible clustering options than clusters in GlassFish; it enables an easy way to combine options for dynamic formation of a lightweight cluster suitable in scalable environments with allowing more control over instances and deployments via Deployment Groups.

The Domain Data Grid is powered by Hazelcast shipped with Payara Server and is a new concept compared to GlassFish clustering. It allows dynamic formation of a cluster suitable in scalable environments, without any additional configuration when adding new instances to the data grid. This isn't possible with GlassFish clustering, which requires modifying the domain configuration when adding each new cluster instance.

Deployment Groups are like clusters in GlassFish. However, Deployment Groups are built on top of the Domain Data Grid and thus are powered by Hazelcast. Unlike clusters in GlassFish, which are based on Shoal (itself implementing the GMS protocol) which has been completely removed in Payara Server 5. The configuration and administration commands for clusters in GlassFish are still supported by Payara Server 5 so that it is easy to migrate them. However, 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. The older traditional clusters are deprecated and not used by default for new clustering configurations. They may be removed in future major Payara Server versions. Deployment Groups and their associated administration commands provide a complete replacement.

Domain Data Grid in Payara Server 5

In order to overcome all clustering challenges in GlassFish, 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.

If you use Shoal Clustering in GlassFish, 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. Although this brings additional benefits, it also means that some features supported in GlassFish are not supported in standard open source version of Payara Server. A notable feature which is missing is secure communication among cluster instances over SSL/TLS. This is available to Payara Enterprise customers for an additional license fee for Payara Scales.

In Payara Server 5, **Hazelcast is enabled by default** to power all the clustering options (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. Instances do not have to be manually added to the data grid. The Domain Administration Server (DAS) will detect all running instances and coordinate all corresponding communication between them. The DAS can also display information about all instances in the Data Grid either by using asadmin commands or the Admin Console as on the following picture:

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 are not 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:

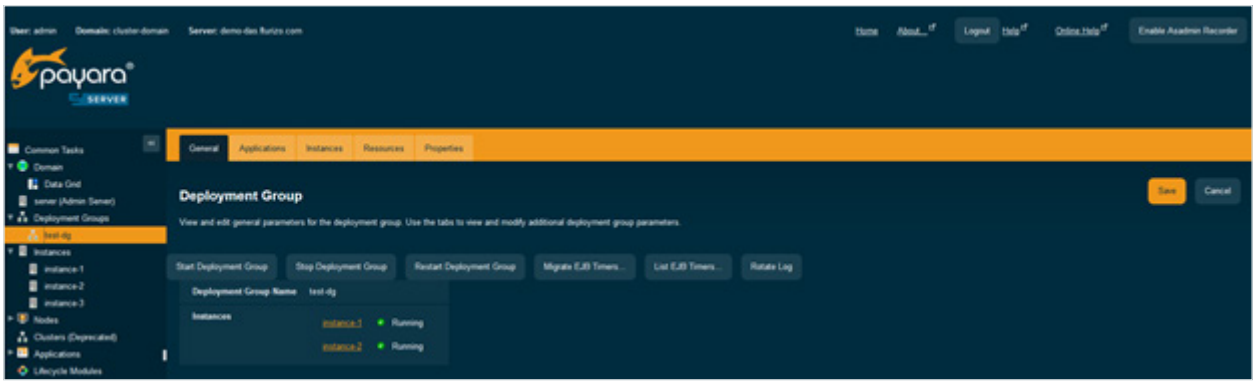
- **DAS**: The Domain Administration Server itself
- **INSTANCE**: Individual instances that are part of the same domain as the DAS or are from a separate domain
- **MICRO**: Payara *Micro* instances that explicitly connect to the domain grid.

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 Data Grid, effectively making them work in a similar vein to Shoal clusters in GlassFish.



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

GlassFish 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 GlassFish 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 GlassFish, standalone instances *can't* be added to a cluster Shoal cluster. *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 GlassFish 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 is 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**:

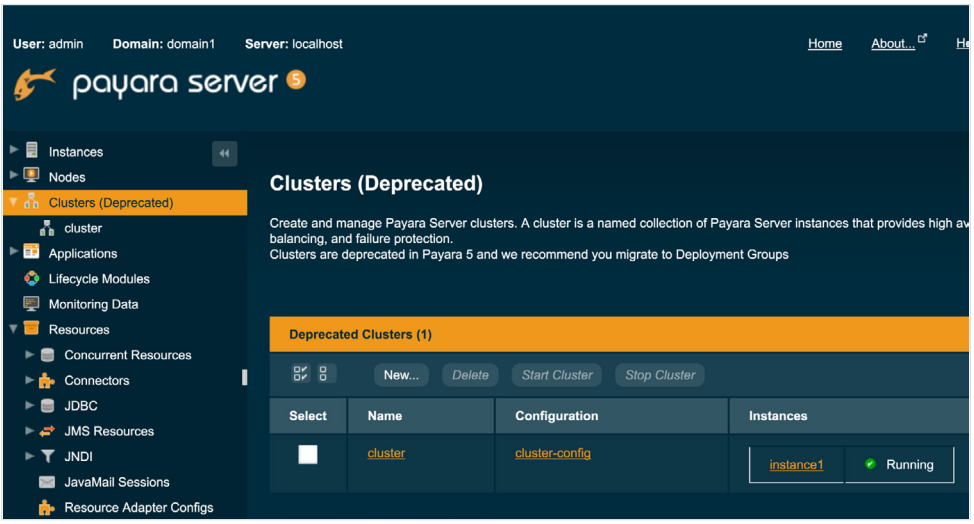
- Instances created directly when creating the Deployment Group
- Instances created separately and added to the Deployment Group

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

- Instances created exclusively for the cluster.

Keeping a Standard GlassFish Cluster

As stated previously, Payara Server 5 will understand the configuration of Shoal Clusters migrated from GlassFish. 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 Domain Data 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 GlassFish. Traditional clusters are now managed in **Clusters (Deprecated)** view on the Admin Console as shown in the image.

If you had established secure communication over SSL/TLS among the instances of your Shoal cluster in GlassFish, 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 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 GlassFish 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 GlassFish 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 on demand. It's also possible to add the same instance to multiple deployment groups.

You can migrate a cluster from GlassFish 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 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 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 CLIcommand	Admin Console	Asadmin CLI 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.	`create-instance`	New instance in the cluster	`create-instance --cluster`
	And existing instance to the group	`add-instance-to-deployment-group`		

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` for example) 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

Mapping Between JSON and Java Objects

Description of the Changes in JSON Mapping

One of the main benefits of Java EE 8 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.

GlassFish (prior to version 5) 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 GlassFIsh, 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 GlassFish and is often used as an alternative to Moxy.

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.

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 continue using the JAX-B annotation configuration on Payara Server 5 as it was on GlassFish. In order 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">
    <context-param>
        <param-name>jersey.config.jsonFeature</param-name>
        <param-value>MoxyJsonFeature</param-value>
    </context-param>
    ...
</web-app>
```

With this configuration, JAX-RS services in your application will support the same mapping with JAX-B annotations as supported in GlassFish. 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, so it is best that, at some point, you'll have 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.

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">
    <context-param>
        <param-name>jersey.config.jsonFeature</param-name>
        <param-value>JacksonFeature</param-value>
    </context-param>
    ...
</web-app>
```

You also needed to have all required Jackson 2 dependencies in your application. This is also required by GlassFish 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 GlassFish. 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.

Migrating from JAX-B Mappings to JSON-B Mappings

Because JSON-B is a standard way of mapping Java objects to JSON payloads it is the best option when building new applications with Payara Server 5. If you are migrating an application from GlassFish, it might not be convenient to refactor all your JAX-B mappings to use it but we still recommend evaluating 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

Built-in Databases

Payara Server 5 includes the **H2 database**, which isn't present in GlassFish. The H2 database is used for the default JDBC data source for applications instead of the Derby DB used in GlassFish. 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. GlassFish 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 by default since it is intended to use by the EJB Timer Service to store persistent timers' information.
- 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 GlassFish

> ***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 GlassFish 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 should not 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.

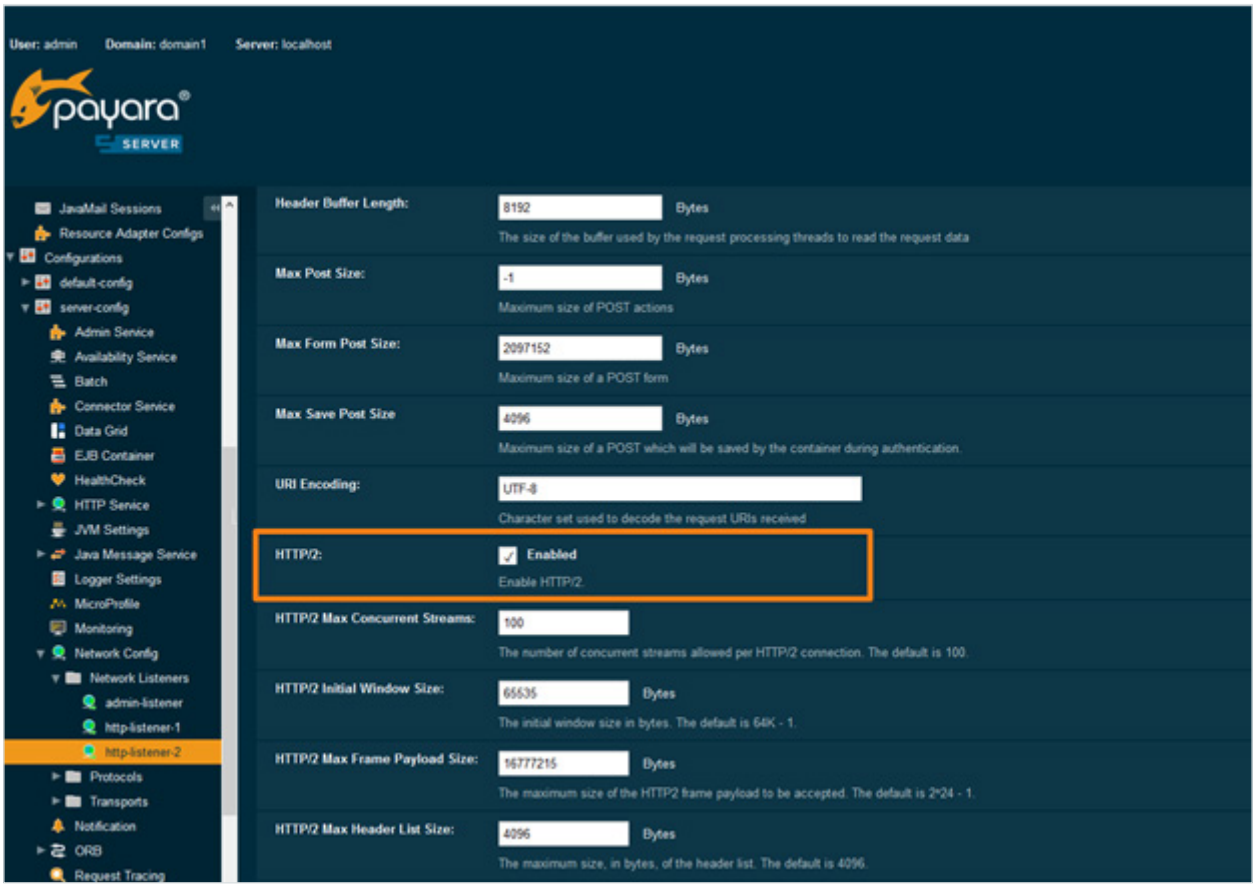
While some web frameworks used for client applications and web browsers can leverage HTTP/2 features to improve overall network performance, HTTP/2 support is not guaranteed to be stable enough in all cases, which could cause issues for your applications. During migration to Payara Server 5, we recommend disabling HTTP/2 support on all HTTP listeners first in order 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 does not 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 HTTP listeners from GlassFish 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
```

How to Replace Features of GlassFish Server Control

The main advantage of using the commercial edition of GlassFish over the Open Source edition is that you could enjoy the benefits of using **GlassFish Server Control**, which is a suite of proprietary features that improves performance, enables fine-grained monitoring and enables more secure and

highly available production deployments. The GlassFish Server Control is composed of the following six features:

- Load Balancer Configurator Plugin
- Domain Administration Server Backup and Recovery
- Coherence Active Cache
- Monitoring Scripting Client
- Oracle Access Management Integration
- Performance Tuner

Since the GlassFish Server Control is only available for the commercial version of GlassFish; these features are licensed separately to the core GlassFish so Payara Server doesn't include these features.

If you use these features intensively, the following sections describe tools and techniques available that can reproduce the functionality of these features and **achieve the same objectives**.

Coherence Active Cache

GlassFish can integrate with Oracle Coherence (an in-memory data grid caching solution) as a replacement for the default in-memory HTTP state replication provided by Shoal.

Payara Server ships with Hazelcast (another in-memory data grid) out-of-the box. It is preconfigured and used as the clustering method. More information on the versatility of Hazelcast and related features like Domain Data Grid, Deployment Groups and JCache, which build on top of Hazelcast, can be found in the Payara Server documentation.

Monitoring Scripting Client

With the Monitoring Scripting Client, operations staff can write JavaScript scripts that enable monitoring probes that help track application performance characteristics, troubleshoot functional issues, follow the state of internal components and services and observe the application behavior in general.

Payara Server, on the other hand, brings multiple monitoring tools to the table:

- An embedded JMX Server with support for AMX MBeans that can be used by users to monitor specific JVM and server statistics in real-time by plugging a compatible JMX console client (JVisualVM, Zulu Mission Control, etc.)
- MicroProfile Metrics support. Thanks to the Metrics API, users can implement automated real-time metrics generation on their application code that can be consumed by modern metric aggregation software tools like Prometheus. Additionally, Payara Server allows

automatic exposure of any JMX/AMX MBean properties as MicroProfile vendor-scoped metrics, easing the effort that the user must put in configurating this aspect.
- A Healthcheck service that can monitor the status of basic environment resources (CPU, RAM, heap size, disk space, etc.) and notify users when certain thresholds are met.
- A customizable monitoring console that can be configured to feed data to the user in real time and is powerful enough to aggregate data from multiple Payara Server nodes and show them in graphic way via the Web console. This is one of the newer features of the Payara Platform and it is intended to become a full featured production solution for Payara Enterprise production environments.

Oracle Access Management Integration

GlassFish Server includes a special security provider, implemented as a custom JASPIC (JSR-196, Java Authentication Service Provider Interface for Containers) module that allows enterprise applications to authenticate and take advantage of the Single Sign On functionality provided by its integration with Oracle Access Manager.

Since Oracle Access Manager is a proprietary product, there is no special integration provided for it, however Payara Server itself contains a simple Single Sign On solution that can be easily configured using standard JAAS mechanisms. When SSO is enabled, all web applications deployed on the same virtual server will share authentication state, so if a user logs in to a web application, they will be implicitly logged for all other remaining applications that require the same authentication.

Performance Tuner

The GlassFish Performance Tuner is a special tool that can be used to automatically tune in the settings of a standalone instance or cluster configuration group by answering a series of questions. GlassFish will take your answers and suggest settings to you and give you the option of either applying all changes directly to the configuration or using the provided instructions to manually apply these changes.

There is, at the time of writing, no replacement feature for the performance tuner plugin however, since it is designed to be used in a one-off way to tune your domain for production, we have provided a new bundled domain and domain template called production which has some sensible defaults already tuned with the common confitions of a production environment in mind.

Load Balancer Configurator Plugin

The Load Balancer plugin is a utility which integrates with a web server (Oracle iPlanet Web Server, Oracle HTTP Server, Apache Web Server and Microsoft IIS) with GlassFish such that configuration can be managed from GlassFish rather than from the web server itself.

There is no replacement feature for this in Payara Server at the time of writing. We recommend to manually setup a load balancer using Apache Web Server with the ***mod_jk*** Tomcat connector or configure a *Nginx* web server with sticky sessions using its available plugins.

Domain Administration Server Backup and Recovery

Both the commercial and open source editions of GlassFish can backup and restore domains. The added value of the GlassFish plug-in is that it allows you to schedule when backups take place natively within GlassFish. A further benefit is that these scheduled backups do not necessitate stopping the domain while the backup takes place.

There is no replacement feature for this in Payara Server, but we plan to add similar features to make it easier to backup, recover and upgrade domains on demand as part of Payara Enterprise.

Features to Consider During or After Migration

There are lots of new features added in Payara Server and not available in GlassFish which can help take your application into the future after you have successfully migrated.

Slow SQL Logging

A crucial production feature which allows you to easily detect when a query to the database exceeds a specific time. This enables you to drill down to the actual line of code impacting production performance enabling rapid triage and fix of production performance issues in the database or inefficient SQL code in your Java EE applications. Slow SQL logging is enabled for a specific datasource in the Admin Console in the advanced properties. When a query exceeds the configured threshold, a WARNING is output into the server log along with a full stack trace of the code that invoked the SQL, allowing rapid identification of the offending code.

Payara Health Check Service

Another powerful tool that makes it easier for the Operation Teams to run Payara Server in production by periodically checking Host CPU Usage; Host Memory Usage; Payara Server's JVM Garbage Collections; Payara Server's JVM Heap Usage; CPU Usage of individual threads. If there is a problem with any of these metrics and they exceed a configurable threshold then a Warning, Error or Critical message is logged to the server's log file, enabling operations teams to rapidly detect problems or work out what happened after problems have occurred.

Request Tracing Service

Ideal tool for developers, it helps you to identify performance issues and their causes to successfully solve them. It allows you to trace requests through the server. All the following request types are traced when the service is enabled: REST (JAX-RS endpoints); Servlet (handling HTTP requests); SOAP Web Service Requests; WebSocket; execution of EJB timers; inbound JMS message handled by a message-driven bean; JBatch job is created; a new task is executed in a managed executor. Request Tracing comes with full Asadmin CLI and Admin Console integration, so you shouldn't have to go hacking around in the *domain.xml* configuration file. You can read more about the Request Tracing Service in Payara Server documentation.

Working with Third-Party Libraries

A significant source of pain for both users of GlassFish 3.x and 4.x is conflicting 3rd party libraries, which may be found in other frameworks like Spring or Grails (based on Spring). Conflicts can arise when you want to use a specific version of a library, but a different version is already included in Payara Server. A common example (until recently) was that Weld shipped an old version of Guava. Since Weld provides the CDI implementation for Payara Server, any application which included Guava would conflict with the server's version and problems would arise.

Payara Server now includes an isolated classloader for both EAR and WAR deployments, so that 3rd party libraries packaged with the application are preferred over those from the server.

There is also **enhanced control over implicit CDI scanning**. By default, as part of the specification, deployments are scanned for any CDI beans which are recognised by having "Bean Defining Annotations". Sometimes, this scanning can cause unwanted effects due to the annotations in 3rd-party libraries triggering the scan, so Payara Server now provides ways to disable implicit CDI scanning from a deployment and to explicitly disable scanning of specific JARs.

Payara Micro

Payara Micro is our microservice-oriented product and works a bit differently to the traditional application server. It enables you to run war files from the command line without any application server installation. Payara Micro is small, <80 MB in size and incredibly simple to use. With its automatic

and elastic clustering, Payara Micro is designed for running enterprise Java applications in a modern containerized/ virtualized infrastructure.

Using the Hazelcast integration each Payara Micro process will automagically cluster with other Payara Micro processes on the network, giving web session resilience and a fully distributed data cache using Payara's JCache support. Payara Micro also comes with a Java API so it can be embedded and launched from your own Java applications - see how.

Cloud Deployment Improvements

One of the main disadvantages of GlassFish clustering is that they are not convenient and 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 clustering includes several features in the form of better clustering integration with cloud environments and friendly 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 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.

Default Role/Group Mapping

GlassFish already can set default group to role mapping in the security configuration for the server, but there is no portable option that can be set in the deployment descriptors. Payara Server introduces an additional setting for deployment descriptors to explicitly enable or disable the default role mapping. This will mean that more configuration necessary for your application can be maintained within the deployment itself, rather than in the application server.

Other Production Features

Payara Server has **a full web-based administration console (Admin Console)** which is fully featured and provides a single view of all clustered and standalone Payara servers. It also has **a fully scriptable Command Line Interface** for the administration of a Payara domain and a full REST based management console. Payara Server provides **a full set of monitoring JMX MBeans** enabling simple and rapid integration to any JMX based monitoring program to provide historical metrics and alerts. And on top of that, Payara Server **supports rolling upgrades of Java/Jakarta EE applications** and can be **fully supported in production** and development 24/7 with 1-hour response time for priority one issues.

Known Issues After Migrating

There are a few known issues now 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
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.

How is Payara Server Better than GlassFish?

- **Always open source and production ready**. Download a production-ready version of Payara Server or Payara Micro.
- **Get support directly from Engineers**. No outsourced helpdesk.
- **Worldwide adoption**. Trusted by global companies, including some from the Fortune 500 list.
- **Enjoy a 10-year software lifecycle**. No need to upgrade a year or two after you implement a Payara solution.
- **Monthly releases, bug fixes, and patches.** Rolled into Payara Enterprise products making Payara Server the best option for production Jakarta EE (Java EE) applications and Payara Micro the best option for containerized Jakarta EE applications.
- **Security alerts and fixes**. Receive notification of security issues and fixes in all versions of the Payara Platform.
- **Docker support** for rapid deployment of virtualised Java EE applications.
- **Vibrant community** and high activity on the Payara GitHub Profile.

Feature	GlassFish 5.x	Payara Server Enterprise 5
License	Open Source	Open Source
Release frequency	Irregular	Monthly
Releases in 2019	1	22: 4 community stream, 12 stability stream, 8 feature stream
Security fixes	Infrequent	• Instant emergency & backported fixes for support customers • As soon as possible for community
Production support	⊗	✓

Feature	GlassFish 5.x	Payara Server Enterprise 5
Migration & Project Support	✗	✓
Component Upgrades (e.g. Tyrus, Mojarra)	Irregular	As needed
Supported IDEs	• Eclipse • Netbeans • IntelliJ IDEA	• Eclipse • Netbeans • IntelliJ IDEA • Visual Studio Code
Caching tools	✗	JCache, Domain Data Grid, Payara Scales (additional cost)
Automatic Clustering	✗	✓ via Hazelcast
Asadmin command recorder	✗	✓
Slow SQL logging	✗	✓
Healthcheck service	✗	✓
Request tracing	✗	✓
Monitoring logging	✗	✓
Microservices distribution	✗	✓ Payara Micro
MicroProfile support	✗	Compatible with MicroProfile
Docker support	Community provided	Official images
HTTP & HTTPS port auto-binding	✗	✓ (Payara Micro only)
Generate Uber JAR	✗	✓ (Payara Micro only)
Production-tuned domain template	✗	✓
Upgrade tool	✗	✓
Jakarta EE Compatible	✓	✓
Embedded Data Grid	✗	✓

Migrating from GlassFish to Payara Server Should Be Relatively Painless!

After following the instructions detailed in this guide you should be able to run your GlassFish 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.

All things considered, migrating from GlassFish to Payara Server should be an easy and painless process. There may be minor hiccups in certain edge-cases, but the majority of the work that may need to be done will be reserved for making use of new features, provided by either Java/Jakarta EE 8 APIs, Payara Server itself, or even Java 8 language features.

If you're still at the early stages, however, the fact that Payara Server is still, operationally, largely similar to GlassFish means that you can "try-before-you-buy" and see how easy it is to swap over simply by using Payara Server to start up your existing domain. It is a simple and easy thing to do and can give you good early insight into just how easy it will be to get started with Payara Server

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, Payara Enterprise customers can enlist the services of Payara Accelerator consultancy. Download our Payara Accelerator Upgrade Guide to learn more about our consultancy solution.

Run the Payara Platform in Production

If you are planning to use Payara Platform in production, make sure to use the fully supported and stable Payara Server Enterprise or Payara Micro Enterprise. Included in your Enterprise subscription:

Choice of support options:

- Migration & Project Support
- 24x7 support
- 10x5 support

sales@payara.fish **+44 207 754 0481** **www.payara.fish**