CHAPTER 3

Setting Up Clusters in GlassFish Server

This chapter describes how to use GlassFish Server clusters. It contains the following sections:

Overview of Clusters

A *cluster* is a named collection of server instances that share the same applications, resources, and configuration information. You can group server instances on different machines into one logical cluster and administer them as one unit. You can easily control the lifecycle of a multi-machine cluster with the DAS.

Instances can be grouped into clusters. You can distribute an application to all instances in the cluster with a single deployment. Clusters are dynamic. When an instance is added or removed, the changes are handled automatically.

Clusters enable horizontal scalability, load balancing, and failover protection. By definition, all the instances in a cluster have the same resource and application configuration. When a server instance or a machine in a cluster fails, the load balancer detects the failure, and redirects traffic from the failed instance to other instances in the cluster. Since the same applications and resources are on all instances in the cluster, an instance can failover to any other instance in the cluster.

Each cluster member sends in-memory state data to another member. As state data is updated in any member, it is replicated. Group Management Service (GMS) can recognize the failure of a member. In that event, the replication framework retrieves the replicated data and notifies members of the failure.

Group Management Service

The Group Management Service (GMS) is an infrastructure component that is enabled for the instances in a cluster. When GMS is enabled, if a clustered instance fails, the cluster and the Domain Administration Server (DAS) are aware of the failure and can take action when failure occurs. Many features of GlassFish Server depend upon GMS. For example, GMS is used by the IIOP failover, in-memory session replication, transaction service, and timer service features.

If server instances in a cluster are located on different machines, ensure that all the server instance machines and the DAS machine are on the same subnet and that multicast is enabled for the network. To test whether multicast is enabled, use the `validate-multicast(1)` subcommand.

GMS is a core service of the Shoal framework. For more information about Shoal, visit the Project Shoal home page (`https://shoal.dev.java.net/`).

The following topics are addressed here:

- "GMS Settings" on page 58
- "To Pre-configure Non-Default GMS Settings" on page 60
- "To Configure GMS Cluster Settings During Cluster Creation" on page 60
- "Configuring GMS Settings Using `asadmin get` and `set`" on page 61
- "To Configure GMS Settings Using the Administration Console" on page 61
- "To Check the Health of Instances in a Cluster" on page 62
- "To Validate that Multicast Transport Is Available for a Cluster" on page 63
- "Using the Multi-Homing Feature With GMS" on page 64

GMS Settings

Some GMS settings are determined during cluster creation. For more information about these settings, see "To Configure GMS Cluster Settings During Cluster Creation" on page 60.

The following settings are used in GMS for group discovery and failure detection:

`group-discovery-timeout-in-millis`
: Indicates the amount of time an instance's GMS module will wait during instance startup for discovering other members of the group.

  The `group-discovery-timeout-in-millis` timeout value should be set to the default or higher. The default is 5000.

`max-missed-heartbeats`
: Indicates the maximum number of missed heartbeats that the health monitor counts before the instance can be marked as a suspected failure. GMS also tries to make a peer-to-peer connection with the suspected member. If the maximum number of missed heartbeats is exceeded and peer-to-peer connection fails, the member is marked as a suspected failure. The default is 3.

`heartbeat-frequency-in-millis`
Indicates the frequency (in milliseconds) at which a heartbeat is sent by each server instance to the cluster.

The failure detection interval is the `max-missed-heartbeats` multiplied by the `heartbeat-frequency-in-millis`. Therefore, the combination of defaults, 3 multiplied by 2000 milliseconds, results in a failure detection interval of 6 seconds.

Lowering the value of `heartbeat-frequency-in-millis` below the default would result in more frequent heartbeat messages being sent out from each member. This could potentially result in more heartbeat messages in the network than a system needs for triggering failure detection protocols. The effect of this varies depending on how quickly the deployment environment needs to have failure detection performed. That is, the (lower) number of retries with a lower heartbeat interval would make it quicker to detect failures.

However, lowering this value could result in false positives because you could potentially detect a member as failed when, in fact, the member's heartbeat is reflecting the network load from other parts of the server. Conversely, a higher timeout interval results in fewer heartbeats in the system because the time interval between heartbeats is longer. As a result, failure detection would take a longer. In addition, a startup by a failed member during this time results in a new join notification but no failure notification, because failure detection and verification were not completed. The lack of a join notification without a preceding failure notification is logged.

The default is 2000.

`verify-failure-waittime-in-millis`
Indicates the verify suspect protocol's timeout used by the health monitor. After a member is marked as suspect based on missed heartbeats and a failed peer–to–peer connection check, the verify suspect protocol is activated and waits for the specified timeout to check for any further health state messages received in that time, and to see if a peer-to-peer connection can be made with the suspect member. If not, then the member is marked as failed and a failure notification is sent. The default is 1500.

`verify-failure-connect-timeout-in-millis`
Indicates the time it takes for the GMS to detect a hardware or network failure of a server instance. Be careful not to set this value too low. The smaller this timeout value is, the greater the chance of detecting false failures. That is, the instance has not failed but doesn't respond within the short window of time. The default is 10000.

The heartbeat frequency, maximum missed heartbeats, peer-to-peer connection-based failure detection, watchdog-based failure reporting, and the verify timeouts are all needed to ensure that failure detection is robust and reliable in GlassFish Server.

▼ To Pre-configure Non-Default GMS Settings

You can pre-configure GMS with values different than the defaults without requiring a restart of the DAS and the cluster.

1. **Create a configuration using the `create-config`(1) subcommand. For example:**

```
asadmin> create-config mycfg
```

2. **Set the values for the created configuration's GMS. For example:**

```
asadmin > set configs.config.mycfg.group-management-service.group-discovery-timeout-in-millis=8000
asadmin> set configs.config.mycfg.group-management-service.failure-detection.max-missed-heartbeats=5
```

3. **Create the cluster so it uses the previously created configuration. For example:**

```
asadmin> create-cluster --config mycfg mycluster
```

See Also You can also view the full syntax and options of a subcommand by typing `asadmin --help` *subcommand* at the command line.

▼ To Configure GMS Cluster Settings During Cluster Creation

1. **Ensure that the DAS is running.**

   Remote subcommands require a running server.

2. **Create a cluster by using the `create-cluster(1)` subcommand.**

   Use the `--gmsenabled`, `--multicastport`, `--multicastaddress`, and `--bindaddress` options and the `GMS_LISTENER_PORT`, `GMS_LOOPBACK`, and `GMS_MULTICAST_TIME_TO_LIVE` properties to configure GMS.

Example 3–1 Creating a Cluster and Setting GMS Options

This example creates a cluster named `cluster1` and configures the `gms-multicast-port` and `gms-multicast-address` settings.

```
asadmin> create-cluster --multicastport 2048 --multicastaddress 228.9.3.1
--config mycfg --properties GMS_LOOPBACK=true cluster1
Command create-cluster executed successfully.
```

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

Configuring GMS Settings Using `asadmin` `get` and `set`

Configure GMS for your environment by changing the settings that determine how frequently GMS checks for failures. For example, you can change the heartbeat frequency, maximum missed heartbeats, and so on.

Below are sample `get(1)` subcommands to get all the GMS attributes and properties associated with a cluster named `cluster2`.

```
asadmin> get "clusters.cluster.cluster2.gms*"
clusters.cluster.cluster2.gms-bind-interface-address=${GMS-BIND-INTERFACE-ADDRESS-cluster2}
clusters.cluster.cluster2.gms-enabled=true
clusters.cluster.cluster2.gms-multicast-address=228.9.245.47
clusters.cluster.cluster2.gms-multicast-port=9833
```

```
asadmin> get "clusters.cluster.cluster2.property.*"
clusters.cluster.cluster2.property.GMS_LISTENER_PORT=${GMS_LISTENER_PORT-cluster2}
clusters.cluster.cluster2.property.GMS_MULTICAST_TIME_TO_LIVE=4
clusters.cluster.cluster2.property.GMS_LOOPBACK=false
```

```
asadmin> get "cluster2.group-management-service.*"
cluster2.group-management-service.failure-detection.heartbeat-frequency-in-millis=2000
cluster2.group-management-service.failure-detection.max-missed-heartbeats=3
cluster2.group-management-service.failure-detection.verify-failure-connect-timeout-in-millis=10000
cluster2.group-management-service.failure-detection.verify-failure-waittime-in-millis=1500
cluster2.group-management-service.group-discovery-timeout-in-millis=5000
```

To change a setting, use a `set(1)` subcommand. Below are sample `set` subcommands to set two GMS attributes associated with a cluster named `cluster2`.

```
asadmin> set clusters.cluster.cluster2.gms-multicast-port=9855
```

```
asadmin> set cluster2.group-management-service.group-discovery-timeout-in-millis=6000
```

Configuring the GMS cluster settings during cluster creation is recommended, because changing them using the `set` subcommand requires a DAS and cluster restart. To configure GMS group discovery and failure detection settings without a restart, see "To Pre-configure Non-Default GMS Settings" on page 60.

▼ To Configure GMS Settings Using the Administration Console

1. **In the Administration Console, click Configurations —> New.**
2. **Type a name for the configuration and select OK.**
3. **Click Configurations —>** *config-name* **—> Group Management Service.**

4 **On the Edit Group Management Service page, configure GMS failure detection settings:**

- Maximum Missed Heartbeats
- Heartbeat Frequency
- Group Discovery Timeout
- Failure Verification Wait Time

5 **In the Administration Console, click Clusters —> New.**

6 **Type a name for the cluster.**

7 **Select the configuration you created previously from the drop-down list. You can either copy or reference this configuration.**

8 **You can create server instances for the cluster as part of cluster creation or after cluster creation. To create them as part of cluster creation, select New and type an instance name for each one.**

9 **Select OK.**

10 **Click Clusters —>** *cluster-name***.**

11 **Under General Information, configure cluster-related GMS settings:**

- GMS Enabled
- Multicast Port
- Multicast Address
- Bind Interface Address

12 **Under Properties —> Cluster Properties, configure cluster-related GMS properties:**

- `GMS_LISTENER_PORT`
- `GMS_MULTICAST_TIME_TO_LIVE`
- `GMS_LOOPBACK`

▼ To Check the Health of Instances in a Cluster

The `get-health` subcommand only works when GMS is enabled. This is the quickest way to evaluate the health of a cluster and to detect if cluster is properly operating; that is, all members of the cluster are running and visible to DAS.

If multicast is not enabled for the network, all instances could be running (as shown by the `list-instances`(1) subcommand), yet isolated from each other. The `get-health` subcommand does not show the instances if they are running but cannot discover each other due to multicast not being configured properly. See "To Validate that Multicast Transport Is Available for a Cluster" on page 63.

1 **Ensure that the DAS and cluster are running.**

Remote subcommands require a running server.

2 **Check whether server instances in a cluster are running by using the `get-health(1)` subcommand.**

Example 3–2 Checking the Health of Instances in a Cluster

This example checks the health of a cluster named `cluster1`.

```
asadmin> get-health cluster1
instance1 started since Wed Sep 29 16:32:46 EDT 2010
instance2 started since Wed Sep 29 16:32:45 EDT 2010
Command get-health executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin --help get-health` at the command line.

▼ To Validate that Multicast Transport Is Available for a Cluster

Before You Begin To test a specific multicast address, multicast port, or bind interface address, get this information beforehand using the `set` subcommand. Use the following subcommand to get the multicast address and port for a cluster named `c1`:

```
asadmin> get "clusters.cluster.c1.gms-multicast*"
clusters.cluster.c1.gms-multicast-address=228.9.174.162
clusters.cluster.c1.gms-multicast-port=5383
```

Use the following subcommand to get the bind interface address of a server instance named `i1`that belongs to a cluster named `c1`:

```
asadmin> get servers.server.i1.system-property.GMS-BIND-INTERFACE-ADDRESS-c1
servers.server.i1.system-property.GMS-BIND-INTERFACE-ADDRESS-c1.name=GMS-BIND-INTERFACE-ADDRESS-c1
servers.server.i1.system-property.GMS-BIND-INTERFACE-ADDRESS-c1.value=10.12.152.30
```

● **Check whether multicast transport is available for a cluster by using the `validate-multicast(1)` subcommand.**

Example 3–3 Validating that Multicast Transport Is Available for a Cluster

This example checks whether multicast transport is available for a cluster named `c1`.

```
asadmin> validate-multicast --multicastaddress=228.9.175.162 --multicastport=5383
Will use port 5,383
Will use address 228.9.175.162
Will use bind interface null
Will use wait period 2,000 (in milliseconds)

Listening for data...
Sending message with content "sr1" every 2,000 milliseconds
Received data from sr1 (loopback)
Exiting after 20 seconds. To change this timeout, use the --timeout command line option.
Command validate-multicast executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin --help get-health` at the command line.

Using the Multi-Homing Feature With GMS

Multi-homing enables GlassFish Server clusters to be used in an environment that uses multiple Network Interface Cards (NICs). A multi-homed host has multiple network connections, of which the connections may or may not be the same network. Multi-homing provides the following benefits:

- Provides redundant network connections within the same subnet. Having multiple NICs ensures that one or more network connections are available for communication.
- Supports communication across two or more different subnets. The DAS and all server instances in the same cluster must be on the same subnet for GMS communication, however.
- Binds to a specific IPv4 address and receives GMS messages in a system that has multiple IP addresses configured. The responses for GMS messages received on a particular interface will also go out through that interface.
- Supports separation of external and internal traffic.

Traffic Separation Using Multi-Homing

You can separate the internal traffic resulting from GMS from the external traffic. Traffic separation enables you plan a network better and augment certain parts of the network, as required.

Consider a simple cluster, `c1`, with three instances, `i101`, `i102`, and `i103`. Each instance runs on a different machine. In order to separate the traffic, the multi-homed machine should have at least two IP addresses belonging to different networks. The first IP as the external IP and the second one as internal IP. The objective is to expose the external IP to user requests, so that all the traffic from the user requests would be through them. The internal IP is used only by the cluster instances for internal communication through GMS. The following procedure describes how to set up traffic separation.

To configure multi-homed machines for GMS without traffic separation, skip the steps or commands that configure the `EXTERNAL-ADDR` system property, but perform the others.

To avoid having to restart the DAS or cluster, perform the following steps in the specified order.

▼ To Set Up Traffic Separation

1 Create the system properties `EXTERNAL-ADDR` and `GMS-BIND-INTERFACE-ADDRESS-c1` for the DAS.

- `asadmin create-system-properties --target server EXTERNAL-ADDR=0.0.0.0`
- `asadmin create-system-properties --target server GMS-BIND-INTERFACE-ADDRESS-c1=0.0.0.0`

2 Create the cluster with the default settings.

Use the following command:

```
asadmin create-cluster c1
```

A reference to a system property for GMS traffic is already set up by default in the `gms-bind-interface-address` cluster setting. The default value of this setting is `${GMS-BIND-INTERFACE-ADDRESS-`*cluster-name*`}`.

3 When creating the clustered instances, configure the external and GMS IP addresses.

Use the following commands:

- `asadmin create-instance --node localhost --cluster c1 --systemproperties EXTERNAL-ADDR=10.12.152.29:GMS-BIND-INTERFACE-ADDRESS-c1=10.12.152.30 i101`
- `asadmin create-instance --node localhost --cluster c1 --systemproperties EXTERNAL-ADDR=10.12.152.39:GMS-BIND-INTERFACE-ADDRESS-c1=10.12.152.40 i102`
- `asadmin create-instance --node localhost --cluster c1 --systemproperties EXTERNAL-ADDR=10.12.152.49:GMS-BIND-INTERFACE-ADDRESS-c1=10.12.152.50 i103`

4 Set the address attribute of HTTP listeners to refer to the `EXTERNAL-ADDR` system properties.

Use the following commands:

```
asadmin set c1-config.network-config.network-listeners.network-listener.http-1.address=\${EXTERNAL-ADDR}
asadmin set c1-config.network-config.network-listeners.network-listener.http-2.address=\${EXTERNAL-ADDR}
```

Working with Clusters

- "To Delete a Cluster" on page 67
- "To Migrate EJB Timers" on page 67

▼ To Create a Cluster

TBD

● **TBD**

▼ To Add a GlassFish Server Instance to a Cluster

TBD

● **TBD**

▼ To Configure a Cluster

TBD

● **TBD**

▼ To Start, Stop, and Delete Clustered Instances

TBD

● **TBD**

▼ To Configure Server Instances in a Cluster

TBD

● **TBD**

▼ To Configure Applications for a Cluster

TBD

● **TBD**

▼ To Configure Resources for a Cluster

TBD

● **TBD**

▼ To Delete a Cluster

TBD

● **TBD**

▼ To Migrate EJB Timers

If a GlassFish Server server instance stops or fails abnormally, it may be desirable to migrate the EJB timers defined for that stopped server instance to a another running server instance.

The EJB timers can be migrated from the stopped source instance to a specified target instance. If no target instance is specified, the DAS will attempt to find a suitable server instance or multiple server instances. A migration notification will then be sent to the selected target server instances.

Note the following restrictions:

- If the source is a standalone instance, then the target must also be a standalone instance.
- If the source instance is part of a cluster, then the target instance must also be part of that same cluster.
- It is not possible to migrate timers from a standalone instance to a clustered instance, or from one cluster to another cluster.
- All EJB timers defined for a given instance are migrated with this procedure. It is not possible to migrate individual timers.

Before You Begin The server instance from which the EJB timers are to be migrated should not be active during the migration process.

1 **Verify that the source server instance from which the EJB timers are to be migrated is not currently running.**

```
asadmin> list-instances source-instance
```

2 **Stop the instance from which the timers are to be migrated, if that instance is still running.**

```
asadmin> stop-instance source-instance
```

Note – The target instance to which the timers will be migrated does not need to be stopped.

---

3 **(Optional) List the currently defined EJB timers on the source instance, if desired.**

- **If the source is a standalone instance:**

  ```
  asadmin> list-timers source-instance
  ```

- **If the source instance is part of a cluster:**

  ```
  asadmin> list-timers source-cluster
  ```

4 **Migrate the timers from the stopped source instance to the target instance.**

```
asadmin> migrate-timers --target target-instance source-instance
```

5 **Restart the target instance to which the EJB timers have been migrated.**

Example 3–4 Migrating an EJB Timer

The following example show how to migrate timers from a standalone source instance named `football` to a standalone target instance named `soccer`.

```
asadmin> migrate-timers --target soccer football
```

See Also `list-timers(1)`, `migrate-timers(1)`, `list-instances(1)`, `stop-instance(1)`

▼ To Upgrade Components Without Loss of Service

In a clustered environment, a rolling upgrade redeploys an application with a minimal loss of service and sessions. A session can be any replicable artifact, including:

- `HttpSession`
- `SingleSignOn`
- `ServletTimer`
- `DialogFragment`
- stateful session bean

You can use the load balancer and multiple clusters to upgrade components within the GlassFish Server without any loss of service. A component can, for example, be a JVM, the GlassFish Server, or a web application.

A rolling upgrade can take place under light to moderate load conditions. The procedure should be doable in a brief amount of time, about 10-15 minutes per server instance.

Applications must be compatible across the upgrade. They must work correctly during the transition, when some server instances are running the old version and others the new one. The old and new versions must have the same shape (for example, non-transient instance variables) of `Serializable` classes that form object graphs stored in sessions. Or, if the shape of these classes is changed, then the application developer must ensure that correct `Serialization` behavior occurs. If the application is not compatible across the upgrade, the cluster must be stopped for a full redeployment.

This approach is not possible if:

- You change the schema of the high-availability database (HADB). For more information, see **Broken Link (Target ID: ABDDS)**

  Note – The HADB software is supplied with the GlassFish Server standalone distribution of Oracle GlassFish Server. For information about available distributions of Oracle GlassFish Server, see "Distribution Types and Their Components" in *Sun Java System Application Server 9.1 Installation Guide*. HADB features are available only in the enterprise profile. For information about profiles, see "Usage Profiles" in *Sun Java System Application Server 9.1 Administration Guide*.

- You perform an application upgrade that involves a change to the application database schema.

Caution – Upgrade all server instances in a cluster together. Otherwise, there is a risk of version mismatch caused by a session failing over from one instance to another where the instances have different versions of components running.

---

1 **Stop one of the clusters using the Stop Cluster button on the General Information page for the cluster.**

2 **Upgrade the component in that cluster.**

3 **Start the cluster using the Start Cluster button on the General Information page for the cluster.**

4 **Repeat the process with the other clusters, one by one.**

Because sessions in one cluster will never fail over to sessions in another cluster, there is no risk of version mismatch caused by a session's failing over from a server instance that is running one version of the component to another server instance (in a different cluster) that is running a different version of the component. A cluster in this way acts as a safe boundary for session failover for the server instances within it.