◆ ◆ ◆ CHAPTER 9

# Configuring High Availability Session Persistence and Failover

This chapter explains how to enable and configure high availability session persistence.



## Overview of Session Persistence and Failover

GlassFish Server provides high availability session persistence through *failover* of HTTP session data and stateful session bean (SFSB) session data. Failover means that in the event of an server instance or hardware failure, another server instance in a cluster takes over a distributed session.

For example, Java EE applications typically have significant amounts of session state data. A web shopping cart is the classic example of session state. Also, an application can cache frequently-needed data in the session object. In fact, almost all applications with significant user interactions need to maintain session state.

The following topics are addressed here:



### Requirements

A distributed session can run in multiple Oracle GlassFish Server instances, if:

- Each server instance has access to the same session state data. GlassFish Server supports in-memory session replication on other servers in the cluster for maintaining HTTP session and stateful session bean data. In-memory session replication is enabled by default for GlassFish Server clustered environments if the Group Management Service is enabled.

  The use of in-memory replication requires the Group Management Service (GMS) to be enabled. For more information about GMS, see "Group Management Service" on page 64.

  If server instances in a cluster are located on different hosts, ensure that the following prerequisites are met:

  - To ensure that GMS and in-memory replication function correctly, the hosts must be on the same subnet.
  - To ensure that in-memory replication functions correctly, the system clocks on all hosts in the cluster must be synchronized as closely as possible.

  ---

  **Note –** GlassFish Server 3.1 does not support High Availability Database (HADB) configurations. Instead, use in-memory replication, as described in "High Availability Session Persistence" on page 18.

  ---

- Each server instance has the same distributable web application deployed to it. The `web-app` element of the `web.xml` deployment descriptor file must contain the `distributable` element.
- The web application uses high-availability session persistence. If a non-distributable web application is configured to use high-availability session persistence, the server writes an error to the log file.
- The web application must be deployed using the `deploy` or `deploydir` subcommand with the `--availabilityenabled` option set to `true`. For more information on these subcommands, see `deploy`(1) and `deploydir`(1).

# Restrictions

When configuring session persistence and failover, note the following restrictions:

- When a session fails over, any references to open files or network connections are lost. Applications must be coded with this restriction in mind.
- The high availability session persistence service is not compatible with dynamic deployment, dynamic reloading, and autodeployment. These features are for development, not production environments, so you must disable them before enabling the session persistence service. For information about how to disable these features, see the *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

- GlassFish Server 3.1 does not support High Availability Database (HADB) configurations. Instead, use in-memory replication, as described in "High Availability Session Persistence" on page 18.
- You can only bind certain objects to distributed sessions that support failover. Contrary to the Servlet 2.4 specification, GlassFish Server 3.1 does not throw an `IllegalArgumentException` if an object type not supported for failover is bound into a distributed session.

  You can bind the following objects into a distributed session that supports failover:

  - Local home and object references for all EJB components.
  - Colocated stateless session, stateful session, or entity bean reference .
  - Distributed stateless session, stateful session, or entity bean reference.
  - JNDI Context for `InitialContext` and `java:comp/env`.
  - `UserTransaction` objects. However, if the instance that fails is never restarted, any prepared global transactions are lost and might not be correctly rolled back or committed.
  - Serializable Java types.
- You cannot bind the following object types into sessions that support failover:
  - JDBC DataSource
  - Java Message Service (JMS) `ConnectionFactory` and `Destination` objects
  - JavaMail™ Session
  - Connection Factory
  - Administered Objects
  - Web service reference

  In general, for these objects, failover will not work. However, failover might work in some cases, if for example the object is serializable.

## Scope

The availability service can be enabled for the following scopes, ranging from highest to lowest:

- Cluster
- Standalone server instance (not part of a cluster)
- Web, EJB, or JMS container in a cluster
- Application
- Standalone Web, EJB, or JMS module
- Individual Stateful Session Bean (SFSB)

In general, enabling or disabling availability session persistence for a cluster or container involves setting the boolean `availability-service` property to `true` or `false` by means of the

`asadmin set` subcommand. The availability service is enabled by default for GlassFish Server clusters and all Web, EJB, and JMS containers running in a cluster.

The value set for the `availability-service` property is inherited by all child objects running in a given cluster or container unless the value is explicitly overridden at the individual module or application level. For example, if the `availability-service` property is set to `true` for an EJB container, the availability service will be enabled by default for all EJB modules running in that container.

Conversely, to enable availability at a given scope, you must enable it at all higher levels as well. For example, to enable availability at the application level, you must also enable it at the cluster or server instance and container levels.

# Enabling the High Availability Session Persistence Service

This section explains how to configure and enable the high availability session persistence service.



## ▼ To Enable Availability for a Cluster, Standalone Instance or Container

This procedure explains how to enable high availability for a cluster as a whole, or for Web, EJB, or JMS containers that run in a cluster, or for a standalone server instance that is not part of a cluster.

1 **Create a GlassFish Server cluster.**

For more information, see "To Create a Cluster" on page 73.

2 **Set up load balancing for the cluster.**

For instructions, see "Setting Up HTTP Load Balancing" on page 125.

**3 Verify that the cluster and all instances within the cluster for which you want to enable availability is running.**

These steps are also necessary when enabling availability for a Web, EJB, or JMS container running in a cluster. The cluster and all instances in the cluster for which you want to enable availability must be running.

**a. Verify that the cluster is running.**

```
asadmin> list-clusters
```

A list of clusters and their status (running, not running) is displayed. If the cluster for which you want to enable availability is not running, you can start it with the following command:

```
asadmin> start-cluster cluster-name
```

**b. Verify that all instances in the cluster are running.**

```
asadmin> list-instances
```

A list of instances and their status is displayed. If the instances for which you want to enable availability are not running, you can start them by using the following command for each instance:

```
asadmin> start-instance instance-name
```

**4 Use one of the following `asadmin set`(1) subcommands to enable availability for a specific cluster, or for a specific Web, EJB, or JMS container.**

- **For a cluster as a whole**

  ```
  asadmin> set cluster-name-config.availability-service.availability-enabled=true
  ```

  For example, for a cluster named `c1`:

  ```
  asadmin> set c1-config.availability-service.availability-enabled=true
  ```

- **For the Web container in a cluster**

  ```
  asadmin> set cluster-name-config.availability-service \
  .web-container-availability.availability-enabled=true
  ```

- **For the EJB container in a cluster**

  ```
  asadmin> set cluster-name-config.availability-service \
  .ejb-container-availability.availability-enabled=true
  ```

- **For the JMS container in a cluster**

  ```
  asadmin> set cluster-name-config.availability-service \
  .jms-availability.availability-enabled=true
  ```

- **For a standalone server instance (not part of a cluster)**

  ```
  asadmin> set instance-name-config.availability-service.availability-enabled=true
  ```

5 **Restart the standalone server instance or each server instance in the cluster.**

If the instance is currently serving requests, quiesce the instance before restarting it so that the instance gets enough time to serve the requests it is handling. For more information, see **Broken Link (Target ID: ABDID)**.

6 **Enable availability for any SFSB that requires it.**

Select methods for which checkpointing the session state is necessary. For more information, see "Configuring Availability for an Individual Bean" on page 143.

7 **Make each Web module distributable if you want it to be highly available.**

For more information, see "Web Module Deployment Guidelines" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

8 **(Optional) Enable availability for individual applications, web modules, or EJB modules during deployment.**

See the links below for instructions.

**See Also**
- "Configuring Availability for Individual Web Applications" on page 140
- "Using Single Sign-on with Session Failover" on page 141

# Configuring Availability for Individual Web Applications

To enable and configure availability for an individual web application, edit the application deployment descriptor file, `glassfish-web.xml`. The settings in an application's deployment descriptor override the web container's availability settings.

The `session-manager` element's `persistence-type` attribute determines the type of session persistence an application uses. It must be set to `replicated` to enable high availability session persistence.

## Example

```
<glassfish-web-app> ...
  <session-config>
    <session-manager persistence-type="replicated">
      <manager-properties>
        <property name="persistenceFrequency" value="web-method" />
      </manager-properties>
      <store-properties>
        <property name="persistenceScope" value="session" />
      </store-properties>
    </session-manager> ...
</session-config> ...
```

# Using Single Sign-on with Session Failover

In a single application server instance, once a user is authenticated by an application, the user is not required to re-authenticate individually to other applications running on the same instance. This is called *single sign-on*.

For this feature to continue to work even when an HTTP session fails over to another instance in a cluster, single sign-on information must be persisted using in-memory replication. To persist single sign-on information, first, enable availability for the server instance and the web container, then enable single-sign-on state failover.

You can enable single sign-on state failover by using the `asadmin set` command to set the configuration's `availability-service.web-container-availability.sso-failover-enabled` property to true.

For example, use the `set` command as follows, where `config1` is the configuration name:

```
asadmin> set config1.availability-service.web-container-availability. \
sso-failover-enabled="true"
```

## Single Sign-On Groups

Applications that can be accessed through a single name and password combination constitute a *single sign-on group*. For HTTP sessions corresponding to applications that are part of a single sign-on group, if one of the sessions times out, other sessions are not invalidated and continue to be available. This is because time out of one session should not affect the availability of other sessions.

As a corollary of this behavior, if a session times out and you try to access the corresponding application from the same browser window that was running the session, you are not required to authenticate again. However, a new session is created.

Take the example of a shopping cart application that is a part of a single sign-on group with two other applications. Assume that the session time out value for the other two applications is higher than the session time out value for the shopping cart application. If your session for the shopping cart application times out and you try to run the shopping cart application from the same browser window that was running the session, you are not required to authenticate again. However, the previous shopping cart is lost, and you have to create a new shopping cart. The other two applications continue to run as usual even though the session running the shopping cart application has timed out.

Similarly, suppose a session corresponding to any of the other two applications times out. You are not required to authenticate again while connecting to the application from the same browser window in which you were running the session.

**Note –** This behavior applies only to cases where the session times out. If single sign-on is enabled and you invalidate one of the sessions using `HttpSession.invalidate()`, the sessions for all applications belonging to the single sign-on group are invalidated. If you try to access any application belonging to the single sign-on group, you are required to authenticate again, and a new session is created for the client accessing the application.

# Stateful Session Bean Failover

Stateful session beans (SFSBs) contain client-specific state. There is a one-to-one relationship between clients and the stateful session beans. At creation, the EJB container gives each SFSB a unique session ID that binds it to a client.

An SFSB's state can be saved in a persistent store in case a server instance fails. The state of an SFSB is saved to the persistent store at predefined points in its life cycle. This is called

*checkpointing*. If enabled, checkpointing generally occurs after the bean completes any transaction, even if the transaction rolls back.

However, if an SFSB participates in a bean-managed transaction, the transaction might be committed in the middle of the execution of a bean method. Since the bean's state might be undergoing transition as a result of the method invocation, this is not an appropriate time to checkpoint the bean's state. In this case, the EJB container checkpoints the bean's state at the end of the corresponding method, provided the bean is not in the scope of another transaction when that method ends. If a bean-managed transaction spans across multiple methods, checkpointing is delayed until there is no active transaction at the end of a subsequent method.

The state of an SFSB is not necessarily transactional and might be significantly modified as a result of non-transactional business methods. If this is the case for an SFSB, you can specify a list of checkpointed methods, as described in "Specifying Methods to Be Checkpointed" on page 144

If a distributable web application references an SFSB, and the web application's session fails over, the EJB reference is also failed over.

If an SFSB that uses session persistence is undeployed while the GlassFish Server instance is stopped, the session data in the persistence store might not be cleared. To prevent this, undeploy the SFSB while the GlassFish Server instance is running.

# Configuring Availability for the EJB Container

To enable availability for the EJB container use the `asadmin set` command to set the following three properties for the configuration:

- `availability-service.ejb-container-availability.availability-enabled`

- `availability-service.ejb-container-availability.sfsb-persistence-type`
- `availability-service.ejb-container-availability.sfsb-ha-persistence-type`

For example, if `config1` is the configuration name, use the following commands:

```
asadmin> set --user admin --passwordfile password.txt
--host localhost
--port 4849
config1.availability-service.
ejb-container-availability.availability-enabled="true"

asadmin> set --user admin --passwordfile password.txt --host localhost --port
4849
config1.availability-service.
ejb-container-availability.sfsb-persistence-type="file"
asadmin> set --user admin --passwordfile password.txt
--host localhost
--port 4849
config1.availability-service.
ejb-container-availability.sfsb-ha-persistence-type="replicated"
```

### Configuring the SFSB Session Store When Availability Is Disabled

If availability is disabled, the local file system is used for SFSB state passivation, but not persistence. To change where the SFSB state is stored, change the Session Store Location setting in the EJB container. For information about configuring store properties, see the Admin Console online help.

## Configuring Availability for an Individual Application or EJB Module

You can enable SFSB availability for an individual application or EJB module during deployment:

- If you are deploying with the Admin Console, check the Availability Enabled checkbox.
- If you are deploying using use the `asadmin deploy` or `asadmin deploydir` commands, set the `--availabilityenabled` option to `true`. For more information, see `deploy`(1) and `deploydir`(1).

## Configuring Availability for an Individual Bean

To enable availability and select methods to be checkpointed for an individual SFSB, use the `glassfish-ejb-jar.xml` deployment descriptor file.

To enable high availability session persistence, set `availability-enabled="true"` in the `ejb` element.

**EXAMPLE 9–1** Example of an EJB Deployment Descriptor With Availability Enabled

```
<glassfish-ejb-jar>
    ...
    <enterprise-beans>
        ...
        <ejb availability-enabled="true">
            <ejb-name>MySFSB</ejb-name>
        </ejb>
        ...
    </enterprise-beans>
</glassfish-ejb-jar>
```

# Specifying Methods to Be Checkpointed

If enabled, checkpointing generally occurs after the bean completes any transaction, even if the transaction rolls back. To specify additional optional checkpointing of SFSBs at the end of non-transactional business methods that cause important modifications to the bean's state, use the `checkpoint-at-end-of-method` element in the `ejb` element of the `glassfish-ejb-jar.xml` deployment descriptor file.

The non-transactional methods in the `checkpoint-at-end-of-method` element can be:

- `create()` methods defined in the home interface of the SFSB, if you want to checkpoint the initial state of the SFSB immediately after creation
- For SFSBs using container managed transactions only, methods in the remote interface of the bean marked with the transaction attribute `TX_NOT_SUPPORTED` or `TX_NEVER`
- For SFSBs using bean managed transactions only, methods in which a bean managed transaction is neither started nor committed

  Any other methods mentioned in this list are ignored. At the end of invocation of each of these methods, the EJB container saves the state of the SFSB to persistent store.

---

**Note –** If an SFSB does not participate in any transaction, and if none of its methods are explicitly specified in the `checkpoint-at-end-of-method` element, the bean's state is not checkpointed at all even if `availability-enabled="true"` for this bean.

For better performance, specify a *small* subset of methods. The methods should accomplish a significant amount of work or result in important modification to the bean's state.

---

**EXAMPLE 9–2** Example of EJB Deployment Descriptor Specifying Methods Checkpointing

```
<glassfish-ejb-jar>
    ...
    <enterprise-beans>
        ...
        <ejb availability-enabled="true">
            <ejb-name>ShoppingCartEJB</ejb-name>
```

**EXAMPLE 9–2** Example of EJB Deployment Descriptor Specifying Methods Checkpointing *(Continued)*

```
            <checkpoint-at-end-of-method>
                <method>
                    <method-name>addToCart</method-name>
                </method>
            </checkpoint-at-end-of-method>
        </ejb>
        ...
    </enterprise-beans>
</glassfish-ejb-jar>
```