◆ ◆ ◆ CHAPTER 1

High Availability in GlassFish Server

This chapter describes the high availability features in GlassFish Server Open Source Edition 3.1.

The following topics are addressed here:

Overview of High Availability

High availability applications and services provide their functionality continuously, regardless of hardware and software failures. To make such reliability possible, GlassFish Server provides mechanisms for maintaining *application state data* between clustered GlassFish Server instances. Application state data, such as HTTP session data, stateful EJB sessions, and dynamic cache information, is replicated in real time across server instances. If any one server instance goes down, the session state is available to the next *failover* server, resulting in minimum application downtime and enhanced transactional security.

GlassFish Server provides the following high availability features:

Load Balancing With the Apache `mod_jk` Module

A common load balancing configuration for GlassFish Server 3.1 is to use the Apache HTTPD server as the Web server front-end, and the Apache `mod_jk` module as the connector between the Web Server and GlassFish Server. See "Configuring GlassFish Server with Apache HTTPD Server and `mod_jk`" on page 122 for more information.

High Availability Session Persistence

GlassFish Server provides high availability of HTTP requests and session data (both HTTP session data and stateful session bean data).

Java EE applications typically have significant amounts of session state data. A web shopping cart is the classic example of a 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. Both HTTP sessions and stateful session beans (SFSBs) have session state data.

Preserving session state across server failures can be important to end users. If the GlassFish Server instance hosting the user session experiences a failure, the session state can be recovered, and the session can continue without loss of information. High availability is implemented in GlassFish Server by means of *in-memory session replication* on GlassFish Server instances running in a cluster.

For more information about in-memory session replication in GlassFish Server, see "How GlassFish ServerProvides High Availability" on page 20. For detailed instructions on configuring high availability session persistence, see Chapter 9, "Configuring High Availability Session Persistence and Failover."

High Availability Java Message Service

The Java Message Service (JMS) API is a messaging standard that allows Java EE applications and components to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous. The Sun Java System Message Queue (MQ), which implements JMS, is tightly integrated with GlassFish Server, enabling you to create components that rely on JMS, such as message-driven beans (MDBs).

JMS is made highly available through connection pooling and failover and MQ clustering. For more information, see Chapter 10, "Java Message Service Load Balancing and Failover."

Connection Pooling and Failover

GlassFish Server supports JMS connection pooling and failover. The GlassFish Server pools JMS connections automatically. By default, GlassFish Server selects its primary MQ broker randomly from the specified host list. When failover occurs, MQ transparently transfers the load to another broker and maintains JMS semantics.

For more information about JMS connection pooling and failover, see "Connection Pooling and Failover" on page 161.

MQ Clustering

MQ Enterprise Edition supports multiple interconnected broker instances known as a *broker cluster*. With broker clusters, client connections are distributed across all the brokers in the cluster. Clustering provides horizontal scalability and improves availability.

For more information about MQ clustering, see "Using MQ Clusters with GlassFish Server" on page 163.

RMI-IIOP Load Balancing and Failover

With RMI-IIOP load balancing, IIOP client requests are distributed to different server instances or name servers, which spreads the load evenly across the cluster, providing scalability. IIOP load balancing combined with EJB clustering and availability also provides EJB failover.

When a client performs a JNDI lookup for an object, the Naming Service essentially binds the request to a particular server instance. From then on, all lookup requests made from that client are sent to the same server instance, and thus all `EJBHome` objects will be hosted on the same target server. Any bean references obtained henceforth are also created on the same target host. This effectively provides load balancing, since all clients randomize the list of target servers when performing JNDI lookups. If the target server instance goes down, the lookup or EJB method invocation will failover to another server instance.

IIOP Load balancing and failover happens transparently. No special steps are needed during application deployment. If the GlassFish Server instance on which the application client is deployed participates in a cluster, the GlassFish Server finds all currently active IIOP endpoints in the cluster automatically. However, a client should have at least two endpoints specified for bootstrapping purposes, in case one of the endpoints has failed.

For more information on RMI-IIOP load balancing and failover, see Chapter 11, "RMI-IIOP Load Balancing and Failover."

More Information

For information about planning a high-availability deployment, including assessing hardware requirements, planning network configuration, and selecting a topology, see the *GlassFish Server Open Source Edition 3.1 Deployment Planning Guide*. This manual also provides a high-level introduction to concepts such as:

- GlassFish Server components such as node agents, domains, and clusters
- IIOP load balancing in a cluster
- Message queue failover

For more information about developing applications that take advantage of high availability features, see the *GlassFish Server Open Source Edition 3.1 Application Development Guide*.

Tuning High Availability Servers and Applications

For information on how to configure and tune applications and GlassFish Server for best performance with high availability, see the *Oracle GlassFish Server 3.1 Performance Tuning Guide*, which discusses topics such as:

- Tuning persistence frequency and persistence scope
- Checkpointing stateful session beans
- Configuring the JDBC connection pool
- Session size
- Configuring load balancer for best performance

How GlassFish ServerProvides High Availability

GlassFish Server provides high availability through the following subcomponents and features:

- "Storage for Session State Data" on page 20
- "Highly Available Clusters" on page 21

Storage for Session State Data

Storing session state data enables the session state to be recovered after the failover of a server instance in a cluster. Recovering the session state enables the session to continue without loss of information. 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 implemented in GlassFish Server 3.1 as OSGi module. Internally, the replication module uses a consistent hash algorithm to pick a replica server instance within a cluster of instances. This allows the replication module to easily locate the replica or replicated data when a container needs to retrieve the data. The replication module

allows any Serializable POJOs (Plain Old Java Objects) to be replicated and retrieved in a cluster of GlassFish Server instances. This means that the module can be used to provide availability for any POJOs, and not just HTTP and Stateful EJBs.

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 62.

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.

Highly Available Clusters

A *highly available cluster* integrates a state replication service with clusters and load balancer.

Clusters, Instances, Sessions, and Load Balancing

Clusters, server instances, load balancers, and sessions are related as follows:

- A server instance is not required to be part of a cluster. However, an instance that is not part of a cluster cannot take advantage of high availability through transfer of session state from one instance to other instances.
- The server instances within a cluster can be hosted on one or multiple hosts. You can group server instances across different hosts into a cluster.
- A particular load balancer can forward requests to server instances on multiple clusters. You can use this ability of the load balancer to perform an online upgrade without loss of service. For more information, see "Upgrading in Multiple Clusters" on page 142.
- A single cluster can receive requests from multiple load balancers. If a cluster is served by more than one load balancer, you must configure the cluster in exactly the same way on each load balancer.
- Each session is tied to a particular cluster. Therefore, although you can deploy an application on multiple clusters, session failover will occur only within a single cluster.

The cluster thus acts as a safe boundary for session failover for the server instances within the cluster. You can use the load balancer and upgrade components within the GlassFish Server without loss of service.

SSH for Centralized Cluster Administration

GlassFish Server uses secure shell (SSH) to ensure that clusters that span multiple hosts can be administered centrally. To perform administrative operations on GlassFish Server instances that are remote from the domain administration server (DAS), the DAS must be able to communicate with those instances. If an instance is running, the DAS connects to the running instance directly. For example, when you deploy an application to an instance, the DAS connects to the instance and deploys the application to the instance.

However, the DAS cannot connect to an instance to perform operations on an instance that is not running, such as creating or starting the instance. For these operations, the DAS uses SSH to contact a remote host and administer instances there. SSH provides confidentiality and security for data that is exchanged between the DAS and remote hosts.

Note – The use of SSH to enable centralized administration of remote instances is optional. If SSH is not practicable in your environment, you can administer remote instances locally.

For more information, see Chapter 2, "Setting Up SSH for Centralized Administration."

Recovering from Failures

You can use various techniques to manually recover individual subcomponents.

The following topics are addressed here:

Recovering the Domain Administration Server

Loss of the Domain Administration Server (DAS) affects only administration. GlassFish Server clusters and standalone instances, and the applications deployed to them, continue to run as before, even if the DAS is not reachable

Use any of the following methods to recover the DAS:

- Back up the domain periodically, so you have periodic snapshots. After a hardware failure, re-create the DAS on a new host, as described in "Re-Creating the Domain Administration Server (DAS)" in *GlassFish Server Open Source Edition 3.1 Administration Guide*.
- Put the domain installation and configuration on a shared and robust file system (NFS for example). If the primary DAS host fails, a second host is brought up with the same IP address and will take over with manual intervention or user supplied automation.

- Zip the GlassFish Server installation and domain root directory. Restore it on the new host, assigning it the same network identity.

Recovering GlassFish Server Instances

GlassFish Server provides no tools for backing up and restoring GlassFish Server instances. Therefore, you must use the features of he operating system to perform these tasks.

- To back up your instances, create a ZIP archive of the `nodes` directory.
- To restore your instances, unzip the saved backup on a new host with same host name and IP address. Use the same installation directory location, operating system, and so forth. GlassFish Server must be installed on the host.

Recovering the HTTP Load Balancer and Web Server

There are no explicit commands to back up only a web server configuration. Simply zip the web server installation directory. After failure, unzip the saved backup on a new host with the same network identity. If the new host has a different IP address, update the DNS server or the routers.

Note – This assumes that the web server is either reinstalled or restored from an image first.

The load balancer plugin (plugins directory) and configurations are in the web server installation directory, typically `/opt/SUNWwbsvr`. The *web-install*`/`*web-instance*`/config` directory contains the `loadbalancer.xml` file.

Recovering Message Queue

Message Queue (MQ) configurations and resources are stored in the DAS and can be synchronized to the instances. Any other data and configuration information is in the MQ directories, typically under `/var/imq`, so backup and restore these directories as required. The new host must already contain the MQ installation. Be sure to start the MQ brokers as before when you restore a host.