the Web Server and GlassFish Server. See "Configuring GlassFish Server with Apache HTTP Server and `mod_jk`" on page 126 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 Server Provides High Availability" on page 19. For detailed instructions on configuring high availability session persistence, see Chapter 9, "Configuring High Availability Session Persistence and Failover."

High Availability Java Message Service

GlassFish Server supports the Java Message Service (JMS) API and JMS messaging through its built-in *jmsra* resource adapter communicating with Open Message Queue as the *JMS provider*. This combination is often called the *JMS Service*.

The JMS service makes JMS messaging highly available as follows:

Connection Pooling and Failover

The JMS service pools JMS connections automatically.

By default, the JMS service selects the primary JMS host (Message Queue broker) randomly from the specified JMS host list. When failover occurs, MQ transparently transfers the load to another JMS host in the list, maintains JMS semantics.

For more information about JMS connection pooling and failover, see "Connection Pooling and Failover" on page 147.

Message Queue Broker Clusters

By default, when a GlassFish cluster is created, the JMS service automatically configures a Message Queue broker cluster to provide JMS messaging services, with one clustered broker assigned to each cluster instance. This automatically created broker cluster is configurable to take advantage of the different types of broker clusters supported by Message Queue.

Additionally, Message Queue broker clusters created and managed using Message Queue itself can be used as external, or remote, JMS hosts to provide JMS messaging high availability to both GlassFish standalone instances and clusters.

For more information about Message Queue clustering, see "Using Message Queue Broker Clusters with GlassFish Server" on page 149.

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."

How GlassFish Server Provides 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 20

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 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.

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 132.
- 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.