Eclipse GlassFish Deployment Planning Guide, Release 8

Eclipse GlassFish

Deployment Planning Guide

Release 8

Contributed 2018 - 2024

This book explains how to build a production deployment of Eclipse GlassFish.

Preface

This documentation is part of the Java Enterprise Edition contribution to the Eclipse Foundation and is not intended for use in relation to Java Enterprise Edition or Orace GlassFish. The documentation is in the process of being revised to reflect the new Jakarta EE branding. Additional changes will be made as requirements and procedures evolve for Jakarta EE. Where applicable, references to Jakarta EE or Java Enterprise Edition should be considered references to Jakarta EE.

Please see the Title page for additional license information.

The Deployment Planning Guide explains how to build a production deployment of Eclipse GlassFish.

This preface contains information about and conventions for the entire Eclipse GlassFish (Eclipse GlassFish) documentation set.

Eclipse GlassFish 8 is developed through the GlassFish project open-source community at https://github.com/eclipse-ee4j/glassfish. The GlassFish project provides a structured process for developing the Eclipse GlassFish platform that makes the new features of the Jakarta EE platform available faster, while maintaining the most important feature of Jakarta EE: compatibility. It enables Java developers to access the Eclipse GlassFish source code and to contribute to the development of the Eclipse GlassFish.

Eclipse GlassFish Documentation Set

The Eclipse GlassFish documentation set describes deployment planning and system installation. For an introduction to Eclipse GlassFish, refer to the books in the order in which they are listed in the following table.

Book Title	Description
Release Notes	Provides late-breaking information about the software and the documentation and includes a comprehensive, table-based summary of the supported hardware, operating system, Java Development Kit (JDK), and database drivers.
Quick Start Guide	Explains how to get started with the Eclipse GlassFish product.
Installation Guide	Explains how to install the software and its components.
Upgrade Guide	Explains how to upgrade to the latest version of Eclipse GlassFish. This guide also describes differences between adjacent product releases and configuration options that can result in incompatibility with the product specifications.
Deployment Planning Guide	Explains how to build a production deployment of Eclipse GlassFish that meets the requirements of your system and enterprise.

Book Title	Description
Administration Guide	Explains how to configure, monitor, and manage Eclipse GlassFish subsystems and components from the command line by using the `asadmin`(1M) utility. Instructions for performing these tasks from the Administration Console are provided in the Administration Console online help.
Security Guide	Provides instructions for configuring and administering Eclipse GlassFish security.
Application Deployment Guide	Explains how to assemble and deploy applications to the Eclipse GlassFish and provides information about deployment descriptors.
Application Development Guide	Explains how to create and implement Java Platform, Enterprise Edition (Jakarta EE platform) applications that are intended to run on the Eclipse GlassFish. These applications follow the open Java standards model for Jakarta EE components and application programmer interfaces (APIs). This guide provides information about developer tools, security, and debugging.
Add-On Component Development Guide	Explains how to use published interfaces of Eclipse GlassFish to develop add-on components for Eclipse GlassFish. This document explains how to perform only those tasks that ensure that the add-on component is suitable for Eclipse GlassFish.
Embedded Server Guide	Explains how to run applications in embedded Eclipse GlassFish and to develop applications in which Eclipse GlassFish is embedded.
High Availability Administration Guide	Explains how to configure Eclipse GlassFish to provide higher availability and scalability through failover and load balancing.
Performance Tuning Guide	Explains how to optimize the performance of Eclipse GlassFish.
Troubleshooting Guide	Describes common problems that you might encounter when using Eclipse GlassFish and explains how to solve them.
Error Message Reference	Describes error messages that you might encounter when using Eclipse GlassFish.
Reference Manual	Provides reference information in man page format for Eclipse GlassFish administration commands, utility commands, and related concepts.
Message Queue Release Notes	Describes new features, compatibility issues, and existing bugs for Open Message Queue.
Message Queue Technical Overview	Provides an introduction to the technology, concepts, architecture, capabilities, and features of the Message Queue messaging service.
Message Queue Administration Guide	Explains how to set up and manage a Message Queue messaging system.

Book Title	Description
Message Queue Developer's Guide for JMX Clients	Describes the application programming interface in Message Queue for programmatically configuring and monitoring Message Queue resources in conformance with the Java Management Extensions (JMX).
Message Queue Developer's Guide for Java Clients	Provides information about concepts and procedures for developing Java messaging applications (Java clients) that work with Eclipse GlassFish.
Message Queue Developer's Guide for C Clients	Provides programming and reference information for developers working with Message Queue who want to use the C language binding to the Message Queue messaging service to send, receive, and process Message Queue messages.

Related Documentation

The following tutorials explain how to develop Jakarta EE applications:

- Your First Cup: An Introduction to the Jakarta EE Platform. For beginning Jakarta EE programmers, this short tutorial explains the entire process for developing a simple enterprise application. The sample application is a web application that consists of a component that is based on the Enterprise JavaBeans specification, a JAX-RS web service, and a JavaServer Faces component for the web front end.
- The Jakarta EE Tutorial. This comprehensive tutorial explains how to use Jakarta EE platform technologies and APIs to develop Jakarta EE applications.

Javadoc tool reference documentation for packages that are provided with Eclipse GlassFish is available as follows.

- The Jakarta EE specifications and API specification is located at https://jakarta.ee/specifications/.
- The API specification for Eclipse GlassFish 8, including Jakarta EE platform packages and nonplatform packages that are specific to the Eclipse GlassFish product, is located at https://glassfish.org/docs/.

For information about creating enterprise applications in the NetBeans Integrated Development Environment (IDE), see the NetBeans Documentation, Training & Support page.

For information about the Derby database for use with the Eclipse GlassFish, see the Derby page.

The Jakarta EE Samples project is a collection of sample applications that demonstrate a broad range of Jakarta EE technologies. The Jakarta EE Samples are bundled with the Jakarta EE Software Development Kit (SDK) and are also available from the repository (`https://github.com/eclipse-ee4j/glassfish-samples`).

Typographic Conventions

The following table describes the typographic changes that are used in this book.

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name% su`
		
		`Password:`

AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` filename.
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the User's Guide. A cache is a copy that is stored locally. Do not save the file.

Symbol Conventions

The following table explains symbols that might be used in this book.

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
`-`	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
`+`	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
`>`	Indicates menu item selection in a graphical user interface.	File > New > Templates	From the File menu, choose New. From the New submenu, choose Templates.

Default Paths and File Names

The following table describes the default paths and file names that are used in this book.

Placeholder	Description	Default Value
as-install	Represents the base installation directory for Eclipse GlassFish. In configuration files, as-install is represented as follows: `${com.sun.aas.installRoot}`	• Installations on the Oracle Solaris operating system, Linux operating system, and Mac OS operating system: user's-home-directory`/glassfish8/glassfish` • Installations on the Windows operating system: SystemDrive`:\glassfish8\glassfish`
as-install-parent	Represents the parent of the base installation directory for Eclipse GlassFish.	• Installations on the Oracle Solaris operating system, Linux operating system, and Mac operating system: user's-home-directory`/glassfish8` • Installations on the Windows operating system: SystemDrive`:\glassfish8`
domain-root-dir	Represents the directory in which a domain is created by default.	as-install`/domains/`
domain-dir	Represents the directory in which a domain's configuration is stored. In configuration files, domain-dir is represented as follows: `${com.sun.aas.instanceRoot}`	domain-root-dir/domain-name
instance-dir	Represents the directory for a server instance.	domain-dir/instance-name

1 Product Concepts

Eclipse GlassFish provides a robust platform for the development, deployment, and management of Jakarta EE applications. Key features include scalable transaction management, web services performance, clustering, security, and integration capabilities.

The following topics are addressed here:

- Jakarta EE Platform Overview
- Eclipse GlassFish Components
- Configuration Roadmap for High Availability of Eclipse GlassFish

Jakarta EE Platform Overview

Eclipse GlassFish implements Java platform, Enterprise Edition (Jakarta EE) 7 technology. The Jakarta EE platform is a set of standard specifications that describe application components, APIs, and the runtime containers and services of an application server.

Jakarta EE Applications

Jakarta EE applications are made up of components such as JavaServer Pages (JSP), Java servlets, and Enterprise JavaBeans (EJB) modules. These components enable software developers to build large-scale, distributed applications. Developers package Jakarta EE applications in Java Archive (JAR) files (similar to zip files), which can be distributed to production sites. Administrators install Jakarta EE applications onto Eclipse GlassFish by deploying Jakarta EE JAR files onto one or more server instances (or clusters of instances).

Containers

Each server instance includes two containers: web and EJB. A container is a runtime environment that provides services such as security and transaction management to Jakarta EE components. Web components, such as Java Server Pages and servlets, run within the web container. Enterprise JavaBeans run within the EJB container.

Jakarta EE Services

The Jakarta EE platform provides services for applications, including:

- Naming - A naming and directory service binds objects to names. A Java EE application can locate an object by looking up its Java Naming and Directory Interface (JNDI) name.
- Security - The Java Authorization Contract for Containers (JACC) is a set of security contracts defined for the Jakarta EE containers. Based on the client's identity, containers can restrict access to the container's resources and services.
- Transaction management - A transaction is an indivisible unit of work. For example, transferring funds between bank accounts is a transaction. A transaction management service ensures that a transaction is either completed, or is rolled back.

- Message Service - Applications hosted on separate systems can communicate with each other by exchanging messages using the Java Message Service (JMS). JMS is an integral part of the Jakarta EE platform and simplifies the task of integrating heterogeneous enterprise applications.

Web Services

Clients can access a Jakarta EE application as a remote web service in addition to accessing it through HTTP, RMI/IIOP, and JMS. Web services are implemented using the Java API for XML-based web services (JAX-WS). A Jakarta EE application can also act as a client to web services, which would be typical in network applications.

Web Services Description Language (WSDL) is an XML format that describes web service interfaces. Web service consumers can dynamically parse a WSDL document to determine the operations a web service provides and how to execute them. Eclipse GlassFish distributes web services interface descriptions using a registry that other applications can access through the Java API for XML Registries (JAXR).

Client Access

Clients can access Jakarta EE applications in several ways. Browser clients access web applications using hypertext transfer protocol (HTTP). For secure communication, browsers use the HTTP secure (HTTPS) protocol that uses secure sockets layer (SSL).

Rich client applications running in the Application Client Container can directly lookup and access Enterprise JavaBeans using an Object Request Broker (ORB), Remote Method Invocation (RMI) and the internet inter-ORB protocol (IIOP), or IIOP/SSL (secure IIOP). They can access applications and web services using HTTP/HTTPS, JMS, and JAX-WS. They can use JMS to send messages to and receive messages from applications and message-driven beans.

Clients that conform to the Web Services-Interoperability (WS-I) Basic Profile can access Jakarta EE web services. WS-I is an integral part of the Jakarta EE standard and defines interoperable web services. It enables clients written in any supporting language to access web services deployed to Eclipse GlassFish.

The best access mechanism depends on the specific application and the anticipated volume of traffic. Eclipse GlassFish supports separately configurable listeners for HTTP, HTTPS, JMS, IIOP, and IIOP/SSL. You can set up multiple listeners for each protocol for increased scalability and reliability.

Jakarta EE applications can also act as clients of Jakarta EE components such as Enterprise JavaBeans modules deployed on other servers, and can use any of these access mechanisms.

External Systems and Resources

On the Jakarta EE platform, an external system is called a resource. For example, a database management system is a JDBC resource. Each resource is uniquely identified and by its Java Naming and Directory Interface (JNDI) name. Applications access external systems through the following APIs and components:

- Java Database Connectivity (JDBC) - A database management system (DBMS) provides facilities

for storing, organizing, and retrieving data. Most business applications store data in relational databases, which applications access via JDBC. Eclipse GlassFish includes the Java DB database for use sample applications and application development and prototyping, though it is not suitable for deployment. Eclipse GlassFish provides certified JDBC drivers for connecting to major relational databases. These drivers are suitable for deployment.

- Java Message Service - Messaging is a method of communication between software components or applications. A messaging client sends messages to, and receives messages from, any other client via a messaging provider that implements the Java Messaging Service (JMS) API. Eclipse GlassFish includes a high-performance JMS broker, Open Message Queue.
- Jakarta EE Connectors - The Jakarta EE Connector architecture enables integrating Jakarta EE applications and existing Enterprise Information Systems (EIS). An application accesses an EIS through a portable Jakarta EE component called a connector or resource adapter, analogous to using JDBC driver to access an RDBMS. Resource adapters are distributed as standalone Resource Adapter Archive (RAR) modules or included in Jakarta EE application archives. As RARs, they are deployed like other Jakarta EE components. Eclipse GlassFish includes evaluation resource adapters that integrate with popular EIS.
- Jakarta Mail - Through the Jakarta Mail API, applications can connect to a Simple Mail Transport Protocol (SMTP) server to send email and to an IMAP or POP3 server to receive email.

Eclipse GlassFish Components

This section describes the components in Eclipse GlassFish.

The following topics are addressed here:

- Server Instances
- Administrative Domains
- Clusters
- Named Configurations
- HTTP Load Balancer Plug-in
- IIOP Load Balancing in a Cluster
- Message Queue and JMS Resources

The administration tools, such as the browser-based Administration Console, communicate with the domain administration server (DAS), which in turn communicates with the server instances.

Server Instances

A server instance is a Eclipse GlassFish running in a single Java Virtual Machine (JVM) process. Eclipse GlassFish is certified with Java platform, Standard Edition (Java SE) 11.

It is usually sufficient to create a single server instance on a machine, since Eclipse GlassFish and accompanying JVM are both designed to scale to multiple processors. However, it can be beneficial to create multiple instances on one machine for application isolation and rolling upgrades. In some cases, a large server with multiple instances can be used in more than one administrative domain.

The administration tools makes it easy to create, delete, and manage server instances across multiple machines.

Administrative Domains

An administrative domain (or simply domain) is a group of server instances that are administered together. A server instance belongs to a single administrative domain. The instances in a domain can run on different physical hosts.

You can create multiple domains from one installation of Eclipse GlassFish. By grouping server instances into domains, different organizations and administrators can share a single Eclipse GlassFish installation. Each domain has its own configuration, log files, and application deployment areas that are independent of other domains. Changing the configuration of one domain does not affect the configurations of other domains. Likewise, deploying an application on one domain does not deploy it or make it visible to any other domain.

All hosts in a domain on which the DAS and Eclipse GlassFish instances are running must have the same operating system.

Domain Administration Server (DAS)

A domain has one Domain Administration Server (DAS), a specially designated Eclipse GlassFish instance that hosts the administrative applications. The DAS authenticates the administrator, accepts requests from administration tools, and communicates with server instances in the domain to carry out the requests.

The administration tools are the `asadmin` command-line tool and the browser-based Administration Console. Eclipse GlassFish also provides a RESTful API for server administration. The administrator can view and manage a single domain at a time, thus enforcing secure separation.

The DAS is also sometimes referred to as the admin server or default server. It is referred to as the default server because it is the default target for some administrative operations.

Since the DAS is a Eclipse GlassFish instance, it can also host Jakarta EE applications for testing purposes. However, do not use it to host production applications. You might want to deploy applications to the DAS, for example, if the clusters and instances that will host the production application have not yet been created.

The DAS keeps a repository containing the configuration of its domain and all the deployed applications. If the DAS is inactive or down, there is no impact on the performance or availability of active server instances, however administrative changes cannot be made. In certain cases, for security purposes, it may be useful to intentionally stop the DAS process, for example to reboot the host operating system to install a kernel patch or a hardware upgrade.

Administrative commands are provided to backup and restore the domain configuration and applications. With the standard backup and restore procedures, you can quickly restore working configurations. If the DAS host fails, you must create a new DAS installation to restore the previous domain configuration. For instructions, see "Administering Domains" in Eclipse GlassFish Administration Guide.

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.

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, redirects traffic from the failed instance to other instances in the cluster, and recovers the user session state. Since the same applications and resources are on all instances in the cluster, an instance can failover to any other instance in the cluster.

All hosts in a cluster on which the DAS and Eclipse GlassFish instances are running must have the same operating system.

Clusters, domains, and instances are related as follows:

- An administrative domain can have zero or more clusters.
- A cluster has one or more server instances.
- A cluster belongs to a single domain.

Named Configurations

A named configuration is an abstraction that encapsulates Eclipse GlassFish property settings. Clusters and stand-alone server instances reference a named configuration to get their property settings. With named configurations, Jakarta EE containers' configurations are independent of the physical machine on which they reside, except for particulars such as IP address, port number, and amount of heap memory. Using named configurations provides power and flexibility to Eclipse GlassFish administration.

To apply configuration changes, you simply change the property settings of the named configuration, and all the clusters and stand-alone instances that reference it pick up the changes. You can only delete a named configuration when all references to it have been removed. A domain can contain multiple named configurations.

Eclipse GlassFish comes with a default configuration, called default-config. The default configuration is optimized for developer productivity.

You can create your own named configuration based on the default configuration that you can customize for your own purposes. Use the Administration Console and `asadmin` command line utility to create and manage named configurations.

HTTP Load Balancer Plug-in

The load balancer distributes the workload among multiple physical machines, thereby increasing the overall throughput of the system. The Eclipse GlassFish includes the load balancer plug-ins for

Oracle iPlanet Web Server, Oracle HTTP Server, Apache Web Server, and Microsoft Internet Information Server.

The load balancer plug-in accepts HTTP and HTTPS requests and forwards them to one of the Eclipse GlassFish instances in the cluster. Should an instance fail, become unavailable (due to network faults), or become unresponsive, requests are redirected to existing, available machines. The load balancer can also recognize when a failed instance has recovered and redistribute the load accordingly.

For simple stateless applications, a load-balanced cluster may be sufficient. However, for mission-critical applications with session state, use load balanced clusters with replicated session persistence.

To setup a system with load balancing, in addition to Eclipse GlassFish, you must install a web server and the load-balancer plug-in. Then you must:

- Create Eclipse GlassFish clusters that you want to participate in load balancing.
- Deploy applications to these load-balanced clusters.

Server instances and clusters participating in load balancing have a homogenous environment. Usually that means that the server instances reference the same server configuration, can access the same physical resources, and have the same applications deployed to them. Homogeneity enables configuration consistency, and improves the ability to support a production deployment.

Use the `asadmin` command-line tool to create a load balancer configuration, add references to clusters and server instances to it, enable the clusters for reference by the load balancer, enable applications for load balancing, optionally create a health checker, generate the load balancer configuration file, and finally copy the load balancer configuration file to your web server `config` directory. An administrator can create a script to automate this entire process.

For more details and complete configuration instructions, see "Configuring HTTP Load Balancing" in Eclipse GlassFish High Availability Administration Guide.

Session Persistence

Jakarta 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 a session state. Both HTTP sessions and stateful session beans (SFSBs) have session state data.

While the session state is not as important as the transactional state stored in a database, preserving the session state across server failures can be important to end users. Eclipse GlassFish provides the capability to save, or persist, this session state in a repository. If the Eclipse GlassFish instance that is hosting the user session experiences a failure, the session state can be recovered. The session can continue without loss of information.

Eclipse GlassFish supports the following session persistence types:

- Memory

- Replicated
- File
- Coherence
- Web

With memory persistence, the state is always kept in memory and does not survive failure. With replicated persistence, Eclipse GlassFish uses other server instances in the cluster as the persistence store for both HTTP and SFSB sessions. With file persistence, Eclipse GlassFish serializes session objects and stores them to the file system location specified by session manager properties. For SFSBs, if replicated persistence is not specified, Eclipse GlassFish stores state information in the session-store subdirectory of this location. For more information about Coherence*Web, see Using Coherence*Web with Eclipse GlassFish (`http://docs.oracle.com/cd/E18686_01/coh.37/e18690/glassfish.html`).

Checking an SFSB's state for changes that need to be saved is called checkpointing. When enabled, checkpointing generally occurs after any transaction involving the SFSB is completed, even if the transaction rolls back. For more information on developing stateful session beans, see "Using Session Beans" in Eclipse GlassFish Application Development Guide. For more information on enabling SFSB failover, see "Stateful Session Bean Failover" in Eclipse GlassFish High Availability Administration Guide.

Apart from the number of requests being served by Eclipse GlassFish, the session persistence configuration settings also affect the session information in each request.

For more information on configuring session persistence, see "Configuring High Availability Session Persistence and Failover" in Eclipse GlassFish High Availability Administration Guide.

IIOP Load Balancing in a Cluster

With IIOP load balancing, IIOP client requests are distributed to different server instances or name servers. The goal is to spread the load evenly across the cluster, thus providing scalability. IIOP load balancing combined with EJB clustering and availability features in Eclipse GlassFish provides not only load balancing but also EJB failover.

There are two steps to IIOP failover and load balancing. The first step, bootstrapping, is the process by which the client sets up the initial naming context with one ORB in the cluster. The client attempts to connect to one of the IIOP endpoints. When launching an application client using the `appclient` script, you specify these endpoints using the `-targetserver` option on the command line or `target-server` elements in the `sun-acc.xml` configuration file. The client randomly chooses one of these endpoints and tries to connect to it, trying other endpoints if needed until one works.

The second step concerns sending messages to a specific EJB. By default, all naming look-ups, and therefore all EJB accesses, use the cluster instance chosen during bootstrapping. The client exchanges messages with an EJB through the client ORB and server ORB. As this happens, the server ORB updates the client ORB as servers enter and leave the cluster. Later, if the client loses its connection to the server from the previous step, the client fails over to some other server using its list of currently active members. In particular, this cluster member might have joined the cluster after the client made the initial connection.

When a client performs a JNDI lookup for an object, the Naming Service creates an `InitialContext` (IC) object associated with a particular server instance. From then on, all lookup requests made using that IC object are sent to the same server instance. All `EJBHome` objects looked up with that `InitialContext` are 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 live target servers when creating `InitialContext` objects. If the target server instance goes down, the lookup or EJB method invocation will failover to another server instance.

Adding or deleting new instances to the cluster does not update the existing client's view of the cluster. You must manually update the endpoints list on the client side.

Message Queue and JMS Resources

The Open Message Queue (Message Queue) provides reliable, asynchronous messaging for distributed applications. Message Queue is an enterprise messaging system that implements the Java Message Service (JMS) standard. Message Queue provides messaging for Jakarta EE application components such as message-driven beans (MDBs).

Eclipse GlassFish implements the Java Message Service (JMS) API by integrating Message Queue into Eclipse GlassFish. Eclipse GlassFish includes the Enterprise version of Message Queue which has failover, clustering and load balancing features.

For basic JMS administration tasks, use the Eclipse GlassFish Administration Console and `asadmin` command-line utility.

For advanced tasks, including administering a Message Queue cluster, use the tools provided in the as-install`/mq/bin` directory. For details about administering Message Queue, see the Open Message Queue Administration Guide.

For information on deploying JMS applications and Message Queue clustering for message failover, see Planning Message Queue Broker Deployment.

Configuration Roadmap for High Availability of Eclipse GlassFish

The following procedure lists the major tasks for configuring Eclipse GlassFish for high availability. The procedure also provides cross-references to detailed instructions for performing each task.

To Configure Eclipse GlassFish for High Availability

1. Determine your requirements and goals for performance and QoS.
   For more information, see the following documentation:
   - Establishing Performance Goals
   - Planning the Network Configuration
   - Planning for Availability
2. Size your system.
   For more information, see Design Decisions.

3. Install Eclipse GlassFish and related subcomponents such as a web server.
   For more information, see the following documentation:
   - Eclipse GlassFish Installation Guide
   - Installation guides for related subcomponents, for example, Oracle iPlanet Web Server 7.0.9 Installation and Migration Guide (`http://docs.oracle.com/cd/E19146-01/821-1832/index.html`)
4. If you plan to administer your clusters centrally, set up secure shell (SSH) for centralized administration.
   For more information, see " Setting Up SSH for Centralized Administration" in Eclipse GlassFish High Availability Administration Guide.
5. Configure domains, nodes, clusters, Eclipse GlassFish instances, and virtual servers as required.
   For more information, see the following documentation:
   - "Administering Domains" in Eclipse GlassFish Administration Guide
   - "Administering Eclipse GlassFish Nodes" in Eclipse GlassFish High Availability Administration Guide
   - "Administering Eclipse GlassFish Clusters" in Eclipse GlassFish High Availability Administration Guide
   - "Administering Eclipse GlassFish Instances" in Eclipse GlassFish High Availability Administration Guide
   - "Administering Virtual Servers" in Eclipse GlassFish Administration Guide
6. Configure your load balancer.
   For more information, see "Administering mod_jk" in Eclipse GlassFish Administration Guide.
7. Configure the web container and EJB container for replicated session persistence.
   For more information, see "Configuring High Availability Session Persistence and Failover" in Eclipse GlassFish High Availability Administration Guide.
8. If you are using messaging extensively, configure Java Message Service (JMS) clusters for failover .
   For more information, see the following documentation:
   - Planning Message Queue Broker Deployment
   - "Configuring Java Message Service High Availability" in Eclipse GlassFish High Availability Administration Guide
   - Open Message Queue Administration Guide
9. Deploy applications and configure them for high availability and session failover.
   For more information, see the Eclipse GlassFish Application Deployment Guide.

2 Planning your Deployment

Before deploying Eclipse GlassFish, first determine the performance and availability goals, and then make decisions about the hardware, network, and storage requirements accordingly.

The following topics are addressed here:

- Establishing Performance Goals
- Planning the Network Configuration
- Planning for Availability
- Design Decisions
- Planning Message Queue Broker Deployment

Establishing Performance Goals

At its simplest, high performance means maximizing throughput and reducing response time. Beyond these basic goals, you can establish specific goals by determining the following:

- What types of applications and services are deployed, and how do clients access them?
- Which applications and services need to be highly available?
- Do the applications have session state or are they stateless?
- What request capacity or throughput must the system support?
- How many concurrent users must the system support?
- What is an acceptable average response time for user requests?
- What is the average think time between requests?

You can calculate some of these metrics using a remote browser emulator (RBE) tool, or web site performance and benchmarking software that simulates expected application activity. Typically, RBE and benchmarking products generate concurrent HTTP requests and then report the response time for a given number of requests per minute. You can then use these figures to calculate server activity.

The results of the calculations described in this chapter are not absolute. Treat them as reference points to work against, as you fine-tune the performance of Eclipse GlassFish and your applications.

The following topics are addressed here:

- Estimating Throughput
- Estimating Load on Eclipse GlassFish Instances
- Estimating Bandwidth Requirements
- Estimating Peak Load

Estimating Throughput

In broad terms, throughput measures the amount of work performed by Eclipse GlassFish. For Eclipse GlassFish, throughput can be defined as the number of requests processed per minute per server instance.

As described in the next section, Eclipse GlassFish throughput is a function of many factors, including the nature and size of user requests, number of users, and performance of Eclipse GlassFish instances and back-end databases. You can estimate throughput on a single machine by benchmarking with simulated workloads.

High availability applications incur additional overhead because they periodically save session data. The amount of overhead depends on the amount of data, how frequently it changes, and how often it is saved. The first two factors depend on the application in question; the latter is also affected by server settings.

Estimating Load on Eclipse GlassFish Instances

Consider the following factors to estimate the load on Eclipse GlassFish instances.

The following topics are addressed here:

- Maximum Number of Concurrent Users
- Think Time
- Average Response Time
- Requests Per Minute

Maximum Number of Concurrent Users

Users interact with an application through a client, such as a web browser or Java program. Based on the user's actions, the client periodically sends requests to the Eclipse GlassFish. A user is considered active as long as the user's session has neither expired nor been terminated. When estimating the number of concurrent users, include all active users.

Initially, as the number of users increases, throughput increases correspondingly. However, as the number of concurrent requests increases, server performance begins to saturate, and throughput begins to decline.

Identify the point at which adding concurrent users reduces the number of requests that can be processed per minute. This point indicates when optimal performance is reached and beyond which throughput start to degrade. Generally, strive to operate the system at optimal throughput as much as possible. You might need to add processing power to handle additional load and increase throughput.

Think Time

A user does not submit requests continuously. A user submits a request, the server receives and processes the request, and then returns a result, at which point the user spends some time before submitting a new request. The time between one request and the next is called think time.

Think times are dependent on the type of users. For example, machine-to-machine interaction such as for a web service typically has a lower think time than that of a human user. You may have to consider a mix of machine and human interactions to estimate think time.

Determining the average think time is important. You can use this duration to calculate the number of requests that need to be completed per minute, as well as the number of concurrent users the system can support.

Average Response Time

Response time refers to the amount of time Eclipse GlassFish takes to return the results of a request to the user. The response time is affected by factors such as network bandwidth, number of users, number and type of requests submitted, and average think time.

In this section, response time refers to the mean, or average, response time. Each type of request has its own minimal response time. However, when evaluating system performance, base the analysis on the average response time of all requests.

The faster the response time, the more requests per minute are being processed. However, as the number of users on the system increases, the response time starts to increase as well, even though the number of requests per minute declines.

A system performance graph indicates that after a certain point, requests per minute are inversely proportional to response time. The sharper the decline in requests per minute, the steeper the increase in response time.

The point of the peak load is the point at which requests per minute start to decline. Prior to this point, response time calculations are not necessarily accurate because they do not use peak numbers in the formula. After this point, (because of the inversely proportional relationship between requests per minute and response time), the administrator can more accurately calculate response time using maximum number of users and requests per minute.

Use the following formula to determine $T_{response}$, the response time (in seconds) at peak load:

$T_{response} = n/r - T_{think}$

where

- n is the number of concurrent users
- r is the number requests per second the server receives
- T_{think} is the average think time (in seconds)

  To obtain an accurate response time result, always include think time in the equation.

Example 2-1 Calculation of Response Time

If the following conditions exist:

- Maximum number of concurrent users, n, that the system can support at peak load is 5,000.
- Maximum number of requests, r, the system can process at peak load is 1,000 per second.

Average think time, T_{think}, is three seconds per request.

Thus, the calculation of response time is:

$T_{response} = n/r - T_{think} = (5000/ 1000) - 3 \text{ sec.} = 5 - 3 \text{ sec.}$

Therefore, the response time is two seconds.

After the system's response time has been calculated, particularly at peak load, compare it to the acceptable response time for the application. Response time, along with throughput, is one of the main factors critical to Eclipse GlassFish performance.

Requests Per Minute

If you know the number of concurrent users at any given time, the response time of their requests, and the average user think time, then you can calculate the number of requests per minute. Typically, start by estimating the number of concurrent users that are on the system.

For example, after running web site performance software, the administrator concludes that the average number of concurrent users submitting requests on an online banking web site is 3,000. This number depends on the number of users who have signed up to be members of the online bank, their banking transaction behavior, the time of the day or week they choose to submit requests, and so on.

Therefore, knowing this information enables you to use the requests per minute formula described in this section to calculate how many requests per minute your system can handle for this user base. Since requests per minute and response time become inversely proportional at peak load, decide if fewer requests per minute is acceptable as a trade-off for better response time, or alternatively, if a slower response time is acceptable as a trade-off for more requests per minute.

Experiment with the requests per minute and response time thresholds that are acceptable as a starting point for fine-tuning system performance. Thereafter, decide which areas of the system require adjustment.

Solving for r in the equation in the previous section gives:

$r = n/(T_{response} + T_{think})$

Example 2-2 Calculation of Requests Per Second

For the values:

- n = 2,800 concurrent users
- $T_{response}$ = 1 (one second per request average response time)
- T_{think} = 3, (three seconds average think time)

The calculation for the number of requests per second is:

```
r = 2800 / (1+3) = 700
```

Therefore, the number of requests per second is 700 and the number of requests per minute is 42000.

Planning the Network Configuration

When planning how to integrate the Eclipse GlassFish into the network, estimate the bandwidth requirements and plan the network in such a way that it can meet users' performance requirements.

The following topics are addressed here:

- Setting Up Traffic Separation
- Estimating Bandwidth Requirements
- Calculating Bandwidth Required
- Estimating Peak Load
- Choosing Network Cards
- Identifying Failure Classes

Setting Up Traffic Separation

You can separate external traffic, such as client requests, from the internal traffic, such as session state failover, database transactions, and messaging. Traffic separation enables you to plan a network better and augment certain parts of the network, as required.

To separate the traffic, run each server instance on a multi-homed machine. A multi-homed machine has two IP addresses belonging to different networks, an external IP and an internal IP. The objective is to expose only the external IP to user requests. The internal IP is used only by the cluster instances for internal communication. For details, see "Using the Multi-Homing Feature With GMS" in Eclipse GlassFish High Availability Administration Guide.

To plan for traffic on both networks, see Estimating Bandwidth Requirements. For external networks, follow the guidelines in Calculating Bandwidth Required and Estimating Peak Load. To size the interfaces for internal networks, see Choosing Network Cards.

Estimating Bandwidth Requirements

To decide on the desired size and bandwidth of the network, first determine the network traffic and identify its peak. Check if there is a particular hour, day of the week, or day of the month when overall volume peaks, and then determine the duration of that peak.

During peak load times, the number of packets in the network is at its highest level. In general, if you design for peak load, scale your system with the goal of handling 100 percent of peak volume. Bear in mind, however, that any network behaves unpredictably and that despite your scaling efforts, it might not always be able handle 100 percent of peak volume.

For example, assume that at peak load, five percent of users occasionally do not have immediate network access when accessing applications deployed on Eclipse GlassFish. Of that five percent,

estimate how many users retry access after the first attempt. Again, not all of those users might get through, and of that unsuccessful portion, another percentage will retry. As a result, the peak appears longer because peak use is spread out over time as users continue to attempt access.

Calculating Bandwidth Required

Based on the calculations made in Establishing Performance Goals, determine the additional bandwidth required for deploying Eclipse GlassFish at your site.

Depending on the method of access (T-1 lines, ADSL, cable modem, and so on), calculate the amount of increased bandwidth required to handle your estimated load. For example, suppose your site uses T-1 or higher-speed T-3 lines. Given their bandwidth, estimate how many lines are needed on the network, based on the average number of requests generated per second at your site and the maximum peak load. Calculate these figures using a web site analysis and monitoring tool.

Example 2-3 Calculation of Bandwidth Required

A single T-1 line can handle 1.544 Mbps. Therefore, a network of four T-1 lines can handle approximately 6 Mbps of data. Assuming that the average HTML page sent back to a client is 30 kilobytes (KB), this network of four T-1 lines can handle the following traffic per second:

6,176,000 bits/10 bits = 772,000 bytes per second

772,000 bytes per second/30 KB = approximately 25 concurrent response pages per second.

With traffic of 25 pages per second, this system can handle 90,000 pages per hour (25 x 60 seconds x 60 minutes), and therefore 2,160,000 pages per day maximum, assuming an even load throughout the day. If the maximum peak load is greater than this, increase the bandwidth accordingly.

Estimating Peak Load

Having an even load throughout the day is probably not realistic. You need to determine when the peak load occurs, how long it lasts, and what percentage of the total load is the peak load.

Example 2-4 Calculation of Peak Load

If the peak load lasts for two hours and takes up 30 percent of the total load of 2,160,000 pages, this implies that 648,000 pages must be carried over the T-1 lines during two hours of the day.

Therefore, to accommodate peak load during those two hours, increase the number of T-1 lines according to the following calculations:

648,000 pages/120 minutes = 5,400 pages per minute

5,400 pages per minute/60 seconds = 90 pages per second

If four lines can handle 25 pages per second, then approximately four times that many pages requires four times that many lines, in this case 16 lines. The 16 lines are meant for handling the realistic maximum of a 30 percent peak load. Obviously, the other 70 percent of the load can be handled throughout the rest of the day by these many lines.

Choosing Network Cards

For greater bandwidth and optimal network performance, use at least 100 Mbps Ethernet cards or, preferably, 1 Gbps Ethernet cards between servers hosting Eclipse GlassFish.

Planning for Availability

The following topics are addressed here:

- Rightsizing Availability
- Using Clusters to Improve Availability
- Adding Redundancy to the System

Rightsizing Availability

To plan availability of systems and applications, assess the availability needs of the user groups that access different applications. For example, external fee-paying users and business partners often have higher quality of service (QoS) expectations than internal users. Thus, it may be more acceptable to internal users for an application feature, application, or server to be unavailable than it would be for paying external customers.

There is an increasing cost and complexity to mitigating against decreasingly probable events. At one end of the continuum, a simple load-balanced cluster can tolerate localized application, middleware, and hardware failures. At the other end of the scale, geographically distinct clusters can mitigate against major catastrophes affecting the entire data center.

To realize a good return on investment, it often makes sense to identify availability requirements of features within an application. For example, it may not be acceptable for an insurance quotation system to be unavailable (potentially turning away new business), but brief unavailability of the account management function (where existing customers can view their current coverage) is unlikely to turn away existing customers.

Using Clusters to Improve Availability

At the most basic level, a cluster is a group of Eclipse GlassFish instances—often hosted on multiple physical servers—that appear to clients as a single instance. This provides horizontal scalability as well as higher availability than a single instance on a single machine. This basic level of clustering works in conjunction with the HTTP load balancer plug-in, which accepts HTTP and HTTPS requests and forwards them to one of the instances in the cluster. The ORB and integrated JMS brokers also perform load balancing to Eclipse GlassFish clusters. If an instance fails, becomes unavailable (due to network faults), or becomes unresponsive, requests are redirected only to existing, available machines. The load balancer can also recognize when a failed instance has recovered and redistribute load accordingly.

Adding Redundancy to the System

One way to achieve high availability is to add hardware and software redundancy to the system. When one unit fails, the redundant unit takes over. This is also referred to as fault tolerance. In

general, to maximize high availability, determine and remove every possible point of failure in the system.

Identifying Failure Classes

The level of redundancy is determined by the failure classes (types of failure) that the system needs to tolerate. Some examples of failure classes are:

- System process
- Machine
- Power supply
- Disk
- Network failures
- Building fires or other preventable disasters
- Unpredictable natural catastrophes

Duplicated system processes tolerate single system process failures, as well as single machine failures. Attaching the duplicated mirrored (paired) machines to different power supplies tolerates single power failures. By keeping the mirrored machines in separate buildings, a single-building fire can be tolerated. By keeping them in separate geographical locations, natural catastrophes like earthquakes can be tolerated.

Planning Failover Capacity

Failover capacity planning implies deciding how many additional servers and processes you need to add to the Eclipse GlassFish deployment so that in the event of a server or process failure, the system can seamlessly recover data and continue processing. If your system gets overloaded, a process or server failure might result, causing response time degradation or even total loss of service. Preparing for such an occurrence is critical to successful deployment.

To maintain capacity, especially at peak loads, add spare machines running Eclipse GlassFish instances to the existing deployment.

For example, consider a system with two machines running one Eclipse GlassFish instance each. Together, these machines handle a peak load of 300 requests per second. If one of these machines becomes unavailable, the system will be able to handle only 150 requests, assuming an even load distribution between the machines. Therefore, half the requests during peak load will not be served.

Design Decisions

Design decisions include whether you are designing the system for peak or steady-state load, the number of machines in various roles and their sizes, and the size of the administration thread pool.

The following topics are addressed here:

- Designing for Peak or Steady State Load

- System Sizing
- Sizing the Administration Thread Pool

Designing for Peak or Steady State Load

In a typical deployment, there is a difference between steady state and peak workloads:

- If the system is designed to handle peak load, it can sustain the expected maximum load of users and requests without degrading response time. This implies that the system can handle extreme cases of expected system load. If the difference between peak load and steady state load is substantial, designing for peak loads can mean spending money on resources that are often idle.
- If the system is designed to handle steady state load, it does not have all the resources required to handle the expected peak load. Thus, the system has a slower response time when peak load occurs.

How often the system is expected to handle peak load will determine whether you want to design for peak load or for steady state.

If peak load occurs often—say, several times per day—it may be worthwhile to expand capacity to handle it. If the system operates at steady state 90 percent of the time, and at peak only 10 percent of the time, then it may be preferable to deploy a system designed around steady state load. This implies that the system's response time will be slower only 10 percent of the time. Decide if the frequency or duration of time that the system operates at peak justifies the need to add resources to the system.

System Sizing

Based on the load on the Eclipse GlassFish instances and failover requirements, you can determine the number of applications server instances (hosts) needed. Evaluate your environment on the basis of the factors explained in Estimating Load on Eclipse GlassFish Instances to each Eclipse GlassFish instance, although each instance can use more than one Central Processing Unit (CPU).

Sizing the Administration Thread Pool

The default `admin-thread-pool` size of 50 should be adequate for most cluster deployments. If you have unusually large clusters, you may need to increase this thread pool size. In this case, set the `max-thread-pool-size` attribute to the number of instances in your largest cluster, but not larger than the number of incoming synchronization requests that the DAS can handle.

Planning Message Queue Broker Deployment

The Java Message Service (JMS) API is a messaging standard that allows Jakarta EE applications and components to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous. Message Queue, which implements JMS, is integrated with Eclipse GlassFish, enabling you to create components that send and receive JMS messages, including message-driven beans (MDBs).

Message Queue is integrated with Eclipse GlassFish using a resource adapter also known as a connector module. A resource adapter is a Java EE component defined according to the Jakarta EE Connector Architecture (JCA) Specification. This specification defines a standardized way in which application servers such as Eclipse GlassFish can integrate with enterprise information systems such as JMS providers. Eclipse GlassFish includes a resource adapter that integrates with its own JMS provider, Message Queue. To use a different JMS provider, you must obtain and deploy a suitable resource adapter that is designed to integrate with it.

Creating a JMS resource in Eclipse GlassFish using the Administration Console creates a preconfigured connector resource that uses the Message Queue resource adapter. To create JMS Resources that use any other resource adapter (including `GenericJMSRA`), you must create them under the Connectors node in the Administration Console.

In addition to using resource adapter APIs, Eclipse GlassFish uses additional Message Queue APIs to provide better integration with Message Queue. This tight integration enables features such as connector failover, load balancing of outbound connections, and load balancing of inbound messages to MDBs. These features enable you to make messaging traffic fault-tolerant and highly available.

The following topics are addressed here:

- Multi-Broker Clusters
- Configuring Eclipse GlassFish to Use Message Queue Brokers
- Example Deployment Scenarios

Multi-Broker Clusters

Message Queue supports using 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.

A single message broker scales to about eight CPUs and provides sufficient throughput for typical applications. If a broker process fails, it is automatically restarted. However, as the number of clients connected to a broker increases, and as the number of messages being delivered increases, a broker will eventually exceed limitations such as number of file descriptors and memory.

Having multiple brokers in a cluster rather than a single broker enables you to:

- Provide messaging services despite hardware failures on a single machine.
- Minimize downtime while performing system maintenance.
- Accommodate workgroups having different user repositories.
- Deal with firewall restrictions.

Message Queue allows you to create conventional or enhanced broker clusters. Conventional broker clusters offer service availability. Enhanced broker clusters offer both service and data availability. For more information, see "Configuring and Managing Broker Clusters" in Open Message Queue Administration Guide.

In a conventional cluster, having multiple brokers does not ensure that transactions in progress at the time of a broker failure will continue on the alternate broker. Although Message Queue reestablishes a failed connection with a different broker in a cluster, transactions owned by the failed broker are not available until it restarts. Except for failed in-progress transactions, user applications can continue on the failed-over connection. Service failover is thus ensured.

In an enhanced cluster, transactions and persistent messages owned by the failed broker are taken over by another running broker in the cluster and non-prepared transactions are rolled back. Data failover is ensured for prepared transactions and persisted messages.

Master Broker and Client Synchronization for Conventional Clusters

In a configuration for a conventional broker cluster, each destination is replicated on all of the brokers in a cluster. Each broker knows about message consumers that are registered for destinations on all other brokers. Each broker can therefore route messages from its own directly-connected message producers to remote message consumers, and deliver messages from remote producers to its own directly-connected consumers.

In a cluster configuration, the broker to which each message producer is directly connected performs the routing for messages sent to it by that producer. Hence, a persistent message is both stored and routed by the message's home broker.

Whenever an administrator creates or destroys a destination on a broker, this information is automatically propagated to all other brokers in a cluster. Similarly, whenever a message consumer is registered with its home broker, or whenever a consumer is disconnected from its home broker—either explicitly or because of a client or network failure, or because its home broker goes down—the relevant information about the consumer is propagated throughout the cluster. In a similar fashion, information about durable subscriptions is also propagated to all brokers in a cluster.

A shared database of cluster change records can be configured as an alternative to using a master broker. For more information, see "Configuring and Managing Broker Clusters" in Open Message Queue Administration Guide and "Using Message Queue Broker Clusters With Eclipse GlassFish" in Eclipse GlassFish High Availability Administration Guide.

Configuring Eclipse GlassFish to Use Message Queue Brokers

By default, Message Queue brokers (JMS hosts) run in the same JVM as the Eclipse GlassFish process. However, Message Queue brokers (JMS hosts) can be configured to run in a separate JVM from the Eclipse GlassFish process. This allows multiple Eclipse GlassFish instances or clusters to share the same set of Message Queue brokers.

The Eclipse GlassFish's Java Message Service represents the connector module (resource adapter) for Message Queue. You can manage the Java Message Service through the Administration Console or the `asadmin` command-line utility.

In Eclipse GlassFish, a JMS host refers to a Message Queue broker. The Eclipse GlassFish's Java Message Service configuration contains a JMS Host List (also called AddressList) that contains all the JMS hosts that will be used.

Java Message Service Type

There are three types of integration between Eclipse GlassFish and Message Queue brokers: embedded, local, and remote. You can set this type attribute on the Administration Console's Java Message Service page.

Embedded Java Message Service

If the Type attribute is EMBEDDED, Eclipse GlassFish and the JMS broker are colocated in the same virtual machine. The JMS Service is started in-process and managed by Eclipse GlassFish. In EMBEDDED mode, JMS operations on stand-alone server instances bypass the networking stack, which leads to performance optimization. The EMBEDDED type is most suitable for stand-alone Eclipse GlassFish instances. EMBEDDED mode is not supported for enhanced broker clusters.

With the EMBEDDED type, use the Start Arguments attribute to specify Message Queue broker startup parameters.

With the EMBEDDED type, make sure the Java heap size is large enough to allow Eclipse GlassFish and Message Queue to run in the same virtual machine.

Local Java Message Service

If the Type attribute is LOCAL, Eclipse GlassFish starts and stops the Message Queue broker. When Eclipse GlassFish starts up, it starts the Message Queue broker specified as the Default JMS host. Likewise, when the Eclipse GlassFish instance shuts down, it shuts down the Message Queue broker. The LOCAL type is most suitable for use with enhanced broker clusters, and for other cases where the administrator prefers the use of separate JVMs.

With the LOCAL type, use the Start Arguments attribute to specify Message Queue broker startup parameters.

Remote Java Message Service

If the Type attribute is REMOTE, Eclipse GlassFish uses an externally configured broker or broker cluster. In this case, you must start and stop Message Queue brokers separately from Eclipse GlassFish, and use Message Queue tools to configure and tune the broker or broker cluster. The REMOTE type is most suitable for brokers running on different machines from the server instances (to share the load among more machines or for higher availability), or for using a different number of brokers and server instances.

With the REMOTE type, you must specify Message Queue broker startup parameters using Message Queue tools. The Start Arguments attribute is ignored.

Managing JMS with the Administration Console

In the Administration Console, you can set JMS properties using the Java Message Service node for a particular configuration. You can set properties such as Reconnect Interval and Reconnect Attempts. For more information, see "Administering the Java Message Service (JMS)" in Eclipse GlassFish Administration Guide.

The JMS Hosts node under the Java Message Service node contains a list of JMS hosts. You can add

and remove hosts from the list. For each host, you can set the host name, port number, and the administration user name and password. By default, the JMS Hosts list contains one Message Queue broker, called "default_JMS_host," that represents the local Message Queue broker integrated with Eclipse GlassFish.

In REMOTE mode, configure the JMS Hosts list to contain all the Message Queue brokers in the cluster. For example, to set up a cluster containing three Message Queue brokers, add a JMS host within the Java Message Service for each one. Message Queue clients use the configuration information in the Java Message Service to communicate with Message Queue broker.

Managing JMS with asadmin

In addition to the Administration Console, you can use the `asadmin` command-line utility to manage the Java Message Service and JMS hosts. Use the following `asadmin` commands:

- Configuring Java Message Service attributes: `asadmin set`
- Managing JMS hosts:
  - `asadmin create-jms-host`
  - `asadmin delete-jms-host`
  - `asadmin list-jms-hosts`
- Managing JMS resources:
  - `asadmin create-jms-resource`
  - `asadmin delete-jms-resource`
  - `asadmin list-jms-resources`

  For more information on these commands, see the Eclipse GlassFish Reference Manual or the corresponding man pages.

Default JMS Host

You can specify the default JMS Host in the Administration Console Java Message Service page. If the Java Message Service type is LOCAL, Eclipse GlassFish starts the default JMS host when the Eclipse GlassFish instance starts. If the Java Message Service type is EMBEDDED, the default JMS host is started lazily when needed.

In REMOTE mode, to use a Message Queue broker cluster, delete the default JMS host, then add all the Message Queue brokers in the cluster as JMS hosts. In this case, the default JMS host becomes the first JMS host in the JMS host list.

You can also explicitly set the default JMS host to one of the JMS hosts. When the Eclipse GlassFish uses a Message Queue cluster, the default JMS host executes Message Queue-specific commands. For example, when a physical destination is created for a Message Queue broker cluster, the default JMS host executes the command to create the physical destinations, but all brokers in the cluster use the physical destination.

Example Deployment Scenarios

To accommodate your messaging needs, modify the Java Message Service and JMS host list to suit your deployment, performance, and availability needs. The following sections describe some typical scenarios.

For best availability, deploy Message Queue brokers and Eclipse GlassFishs on different machines, if messaging needs are not just with Eclipse GlassFish. Another option is to run a Eclipse GlassFish instance and a Message Queue broker instance on each machine until there is sufficient messaging capacity.

Default Deployment

Installing the Eclipse GlassFish automatically creates a domain administration server (DAS). By default, the Java Message Service type for the DAS is EMBEDDED. So, starting the DAS also starts its default Message Queue broker.

Creating a new domain also creates a new broker. By default, when you add a stand-alone server instance or a cluster to the domain, its Java Message Service is configured as EMBEDDED and its default JMS host is the broker started by the DAS.

Using a Message Queue Broker Cluster with a Eclipse GlassFish Cluster

In EMBEDDED or LOCAL mode, when a Eclipse GlassFish is configured, a Message Queue broker cluster is auto-configured with each Eclipse GlassFish instance associated with a Message Queue broker instance.

In REMOTE mode, to configure a Eclipse GlassFish cluster to use a Message Queue broker cluster, add all the Message Queue brokers as JMS hosts in the Eclipse GlassFish's Java Message Service. Any JMS connection factories created and MDBs deployed then uses the JMS configuration specified.

Specifying an Application-Specific Message Queue Broker Cluster

In some cases, an application may need to use a different Message Queue broker cluster than the one used by the Eclipse GlassFish cluster. To do so, use the `AddressList` property of a JMS connection factory or the `activation-config` element in an MDB deployment descriptor to specify the Message Queue broker cluster.

For more information about configuring connection factories, see "Administering JMS Connection Factories and Destinations" in Eclipse GlassFish Administration Guide. For more information about MDBs, see "Using Message-Driven Beans" in Eclipse GlassFish Application Development Guide.

Application Clients

When an application client or standalone application accesses a JMS administered object for the first time, the client JVM retrieves the Java Message Service configuration from the server. Further changes to the JMS service will not be available to the client JVM until it is restarted.

3 Checklist for Deployment

This appendix provides a checklist to get started on evaluation and production with the Eclipse GlassFish.

Checklist

Table 3-1 Checklist

Component/Feature	Description
Application	Determine the following requirements for the application to be deployed. • Required/acceptable response time. • Peak load characteristics. • Necessary persistence scope and frequency. • Session timeout in `web.xml`. • Failover and availability requirements. For more information see the Eclipse GlassFish Performance Tuning Guide.
Hardware	• Have necessary amounts of hard disk space and memory installed. • Use the sizing exercise to identify the requirements for deployment. For more information see the Eclipse GlassFish Release Notes
Operating System	• Ensure that the product is installed on a supported platform. • Ensure that the patch levels are up-to-date and accurate. For more information see the Eclipse GlassFish Release Notes
Network Infrastructure	• Identify single points of failures and address them. • Make sure that the NICs and other network components are correctly configured. • Run `ttcp` benchmark test to determine if the throughput meets the requirements/expected result. • Setup `ssh` based your preference. For more information see the Eclipse GlassFish Installation Guide.
Back-ends and other external data sources	Check with the domain expert or vendor to ensure that these data sources are configured appropriately.

Component/Feature	Description
System Changes/Configuration	• Make sure that changes to `/etc/system` and its equivalent on Linux are completed before running any performance/stress tests. • Make sure the changes to the TCP/IP settings are complete. • By default, the system comes with lots of services pre-configured. Not all of them are required to be running. Turn off services that are not needed to conserve system resources. • On Solaris, use `Setoolkit` to determine the behavior of the system. Resolve any flags that show up. For more information see the Eclipse GlassFish Performance Tuning Guide.
Installation	• Ensure that these servers are not installed on NFS mounted volumes. • Check for enough disk space and RAM when installing Eclipse GlassFish.
Eclipse GlassFish Configuration	• Enable access log rotation. • Choose the right logging level. WARNING is usually appropriate. • Configure Jakarta EE containers using the Administration Console. • Configure HTTP listeners using the Administration Console. • Configure ORB threadpool using the Administration Console. • Ensure that the appropriate persistence scope and frequency are used and they are not overridden underneath in the individual Web/EJB modules. • Ensure that only critical methods in the SFSB are checkpointed. For more information on tuning, see the Eclipse GlassFish Performance Tuning Guide. For more information on configuration, see the Eclipse GlassFish Administration Guide.
Load balancer Configuration	• Make sure the load balancer have network access to the Server. • See documentation of your load balancer and verify that the configuration of the load balancer is correct.

Component/Feature	Description
Configuring usual attributes of load balancing algorithm	• Endpoint - Address or host and port used for checks generating minimal possible load. • Response-time-out-in-seconds - How long the load balancer waits before declaring a Eclipse GlassFish instance unhealthy. Set this value based on the response time of the application. If set too high, the Web Server and load balancer plug-in wait a long time before marking a Eclipse GlassFish instance as unhealthy. If set too low and Eclipse GlassFish's response time crosses this threshold, the instance will be incorrectly marked as unhealthy. • Interval-in-seconds - Time in seconds after which unhealthy instances are checked to find out if they have returned to a healthy state. Too low a value generates extra traffic from the load balancer plug-in to Eclipse GlassFish instances and too high a value delays the routing of requests to the instance that has turned healthy. • Timeout-in-seconds - Duration for a response to be obtained for a health check request. Adjust this value based on the traffic among the systems in the cluster to ensure that the health check succeeds.
Configuring time-outs in Eclipse GlassFish	• Max-wait-time-millis - Wait time to get a connection from the pool before throwing an exception. Default is 6 s. Consider changing this value for highly loaded systems where the size of the data being persisted is greater than 50 KB. • Cache-idle-timeout-in-seconds - Time an EJB is allowed to be idle in the cache before it gets passivated. Applies only to entity beans and stateful session beans. • Removal-timeout-in-seconds - Time that an EJB remains passivated (idle in the backup store). Default value is 60 minutes. Adjust this value based on the need for SFSB failover.
Java Virtual Machine Configuration	• Initially set the minimum and maximum heap sizes to be the same, and at least one GB for each instance. • When running multiple instances of Eclipse GlassFish, consider creating a processor set and bind Eclipse GlassFish to it. This helps in cases where the CMS collector is used to sweep the old generation.