Oracle® Fusion Middleware

Administering Clusters for Oracle WebLogic Server

15c (15.1.1.0.0)
G31433-01
October 2025

Oracle Fusion Middleware Administering Clusters for Oracle WebLogic Server, 15c (15.1.1.0.0)

G31433-01

Primary Author: Oracle Corporation

Contents

5 Failover and Replication in a Cluster

6 Whole Server Migration

7 Service Migration

8 Cluster Architectures

9 Setting Up WebLogic Clusters

10 Dynamic Clusters

11 Configuring and Managing Coherence Clusters

12 Clustering Best Practices

Preface

This documentation describes how to plan, implement and support Oracle WebLogic Server 15c clusters.

Audience

This document is written for application developers and administrators who are developing or deploying Web-based applications on one or more clusters. It also contains information that is useful for business analysts and system architects who are evaluating WebLogic Server or considering the use of WebLogic Server clusters for a particular application.

The topics in this document are primarily relevant to planning, implementing, and supporting a production environment that includes WebLogic Server clusters. Key guidelines for software engineers who design or develop applications that will run on a WebLogic Server cluster are also addressed.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

- Understanding Enterprise JavaBeans in *Developing Jakarta Enterprise Beans Using Deployment Descriptors*.
- Creating and Configuring Web Applications in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

New and Changed WebLogic Server Features

For a comprehensive listing of the new WebLogic Server features introduced in this release, see *What's New in Oracle WebLogic Server*.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Understanding WebLogic Server Clustering

This chapter provides a brief introduction to Oracle WebLogic Server clusters. This chapter includes the following sections:

What Is a WebLogic Server Cluster?

A WebLogic Server cluster consists of multiple WebLogic Server instances running simultaneously and working together to provide increased scalability and reliability.

A cluster appears to clients as a single WebLogic Server instance. The server instances that constitute a cluster can run on the same machine, or a different machines.

You can increase a cluster's capacity by either of the following procedures:

- By adding additional server instances to the cluster on an existing machine.
- By adding machines to the cluster to host the incremental server instances.

Each server instance in a cluster must run the same version of WebLogic Server.

What Are Dynamic Clusters?

Dynamic clusters consist of server instances that can be dynamically scaled up to meet the resource needs of your application. A dynamic cluster uses a single server template to define a configuration for a specified number of generated (dynamic) server instances.

When you create a dynamic cluster, the dynamic servers are preconfigured and automatically generated for you, enabling you to easily scale up the number of server instances in your dynamic cluster when you need additional server capacity. You can start the dynamic servers without manual configuration and add them to the cluster.

See Create a Dynamic Cluster in the *Oracle WebLogic Remote Console Online Help*.

How Does a Cluster Relate to a Domain?

A cluster is part of a particular WebLogic Server domain.

A domain is an interrelated set of WebLogic Server resources that are managed as a unit. It includes one or more WebLogic Server instances, which can be clustered, non-clustered, or a combination of clustered and non-clustered instances. It can include multiple clusters. It also contains the application components deployed in the domain, and the resources and services required by those application components and the server instances in the domain. Examples of the resources and services used by applications and server instances include machine definitions, optional network channels, connectors, and startup classes.

You can use a variety of criteria for organizing WebLogic Server instances into domains. For instance, you might choose to allocate resources to multiple domains based on logical divisions of the hosted application, geographical considerations, or the number or complexity of the resources under management. See *Understanding Domain Configuration for Oracle WebLogic Server*.

In each domain, one WebLogic Server instance acts as the Administration Server—the server instance which configures, manages, and monitors all other server instances and resources in the domain. Each Administration Server manages one domain only. If a domain contains multiple clusters, each cluster in the domain has the same Administration Server.

All server instances in a cluster must reside in the same domain; you cannot "split" a cluster over multiple domains. Similarly, you cannot share a configured resource or subsystem between domains.

Clustered WebLogic Server instances behave similarly to non-clustered instances, except that they provide failover and load balancing. The process and tools used to configure clustered WebLogic Server instances are the same as those used to configure non-clustered instances. However, to achieve the load balancing and failover benefits that clustering enables, you must adhere to certain guidelines for cluster configuration.

To understand how the failover and load balancing mechanisms used in WebLogic Server relate to particular configuration options, see Load Balancing in a Cluster, and Failover and Replication in a Cluster.

Detailed configuration recommendations are included throughout the instructions in Setting up WebLogic Clusters.

What Are the Benefits of Clustering?

A WebLogic Server cluster provides the benefits of scalability and high availability.

- Scalability

  The capacity of an application deployed on a WebLogic Server cluster can be increased dynamically to meet demand. You can add server instances to a cluster without interruption of service. The application continues to run without impact on clients and end users.

- High Availability

  In a WebLogic Server cluster, application processing can continue when a server instance fails. You can cluster application components by deploying them on multiple server instances in the cluster. So, if a server instance on which a component is running fails, another server instance on which that component is deployed can continue application processing.

The choice to cluster WebLogic Server instances is transparent to application developers and clients. However, understanding the technical infrastructure that enables clustering will help programmers and administrators maximize the scalability and availability of their applications.

What Are the Key Capabilities of a Cluster?

The key clustering capabilities that enable scalability and high availability are application failover, server migration, and load balancing.

- Application Failover

  Failover means that when an application component (typically referred to as an "object" in the following sections) doing a particular "job"—some set of processing tasks—becomes unavailable for any reason, a copy of the failed object finishes the job.

  For the new object to take over the failed object:

  – There must be a copy of the failed object available to take over the job.

  - There must be information, available to other objects and the program that manages failover, defining the location and operational status of all objects—so that it can be determined that the first object failed before finishing its job.
  - There must be information, available to other objects and the program that manages failover, about the progress of jobs in process—so that an object taking over an interrupted job knows how much of the job was completed before the first object failed, for example, what data has been changed, and what steps in the process were completed.

  WebLogic Server uses standards-based communication techniques and facilities—including IP sockets and the Java Naming and Directory Interface *(JNDI)*—to share and maintain information about the availability of objects in a cluster. These techniques allow WebLogic Server to determine that an object stopped before finishing its job, and where there is a copy of the object to complete the job that was interrupted.

  > **Note**
  >
  > For backward compatibility with previous versions, WebLogic Server allows you to use multicast for communications between clusters.

  Information about what has been done on a job is called *state*. WebLogic Server maintains information about state using techniques called *session replication* and *replica-aware stubs*. When a particular object unexpectedly stops doing its job, replication techniques enable a copy of the object to pick up where the failed object stopped and finish the job.

- Server Migration

  WebLogic Server supports automatic and manual migration of a clustered server instance from one machine to another. A Managed Server that can be migrated is referred to as a *migratable server*. This feature is designed for environments with requirements for high availability. The server migration capability is useful for:

  - Ensuring uninterrupted availability of *singleton services*—services that must run on only a single server instance at any given time, such as Jakarta Messaging Service (JMS) and the Jakarta Transaction API (JTA) transaction recovery system, when the hosting server instance fails. A Managed Server configured for automatic migration will be automatically migrated to another machine in the event of failure.
  - Easing the process of relocating a Managed Server and all the services it hosts, as part of a planned system administration process. To initiate the migration of a Managed Server, you can use any of the administration tools listed in Summary of System Administration Tools and APIs in *Understanding Oracle WebLogic Server*.

  The server migration process relocates a Managed Server in its entirety—including IP addresses and hosted applications—to one of a predefined set of available host machines.

- Load Balancing

  Load balancing is the even distribution of jobs and associated communications across the computing and networking resources in your environment.

  For load balancing to occur:

  - There must be multiple copies of an object that can do a particular job.
  - WebLogic Server allows objects to be clustered—deployed on multiple server instances—so that there are alternative objects to do the same job. WebLogic Server shares and maintains the availability and location of deployed objects using unicast, IP sockets, and JNDI.

> **Note**
>
> For backward compatibility with previous versions, WebLogic Server also allows you to use multicast for communications between clusters.

Information about the location and operational status of all objects must be available.

A detailed discussion of how communications and replication techniques are employed by WebLogic Server is provided in Communications In a Cluster.

What Types of Objects Can Be Clustered?

A clustered application or application component is one that is available on multiple WebLogic Server instances in a cluster. Failover and load balancing is available for the clustered object. Deploy objects homogeneously to every server instance in your cluster to simplify cluster administration, maintenance, and troubleshooting. Learn the types of objects that can be clustered in a WebLogic Server deployment.

Web applications can consist of different types of objects, including Jakarta Enterprise Beans (EJBs), servlets, and Jakarta Server Pages (JSPs). Each object type has a unique set of behaviors related to control, invocation, and how it functions within an application. For this reason, the methods that WebLogic Server uses to support clustering—and hence to provide load balancing and failover—can vary for different types of objects. The following types of objects can be clustered in a WebLogic Server deployment:

- Servlets
- JSPs
- EJBs
- Remote Method Invocation (RMI) objects
- JMS destinations
- Batch applications

Different object types can have certain behaviors in common. When this is the case, the clustering support and implementation considerations for those similar object types may be same. In the following sections, explanations and instructions for the following types of objects are generally combined:

- Servlets and JSPs
- EJBs and RMI objects

The following sections briefly describe the clustering, failover, and load balancing support that WebLogic Server provides for different types of objects.

Servlets and JSPs

WebLogic Server provides clustering support for servlets and JSPs by replicating the HTTP session state of clients that access clustered servlets and JSPs. WebLogic Server can maintain HTTP session states in memory, a file system, or a database.

To enable automatic failover of servlets and JSPs, session state must persist in memory. For information about how failover works for servlets and JSPs, and for related requirements and programming considerations, see HTTP Session State Replication.

You can balance the servlet and JSP load across a cluster using a WebLogic Server proxy plug-in or external load balancing hardware. WebLogic Server proxy plug-ins perform round-robin load balancing. External load balancers typically support a variety of session load balancing mechanisms. See Load Balancing for Servlets and JSPs.

EJBs and RMI Objects

Load balancing and failover for EJBs and RMI objects is handled using *replica-aware stubs*, which can locate instances of the object throughout the cluster. Replica-aware stubs are created for EJBs and RMI objects as a result of the object compilation process. EJBs and RMI objects are deployed homogeneously—to all the server instances in the cluster.

Failover for EJBs and RMI objects is accomplished using the object's replica-aware stub. When a client makes a call through a replica-aware stub to a service that fails, the stub detects the failure and retries the call on another replica. To understand failover support for different types of objects, see Replication and Failover for EJBs and RMIs.

WebLogic Server clusters support multiple algorithms for load balancing clustered EJBs and RMI objects: round-robin, weight-based, random, round-robin-affinity, weight-based-affinity, and random-affinity. By default, a WebLogic Server cluster will use the round-robin method. You can configure a cluster to use one of the other methods using the WebLogic Remote Console. The method you select is maintained within the replica-aware stub obtained for clustered objects. See Load Balancing for EJBs and RMI Objects for details.

JMS and Clustering

The WebLogic JMS architecture implements clustering of multiple JMS servers by supporting cluster-wide, transparent access to destinations from any WebLogic Server instance in the cluster. Although WebLogic Server supports distributing JMS destinations and connection factories throughout a cluster, the same JMS topic or queue is still managed separately by each WebLogic Server instance in the cluster.

Load balancing is supported for JMS. To enable load balancing, you must configure targets for JMS servers. For more information about load balancing and JMS components, see Load Balancing for JMS. For instructions on setting up clustered JMS, see Configure Migratable Targets for Pinned Services, and Deploying_ Activating_ and Migrating Migratable Services.

Batch Applications

WebLogic Server supports the ability to deploy batch applications in a cluster, and also the ability to have an individual batch runtime instance running on each Managed Server in that cluster. Scaling up the number of servers in the cluster will generally increase the overall batch processing capability of the application deployed to the cluster. However, the processing of any individual batch job is executed on a single managed server within the cluster and cannot be distributed across managed servers in the cluster.

To deploy and run batch applications in a cluster, note the following:

- An individual batch runtime must be configured on each Managed Server instance in the cluster. When a batch job has been started on a Managed Server instance, the job runs on that instance to completion. See Use of Multiple Batch Runtime Instances in *Administering Server Environments for Oracle WebLogic Server* for important considerations regarding the implications of this behavior on load balancing.

- The batch runtime reference implementation (RI) uses a database to model the job repository, which contains details about the past and current jobs. Configuring the job repository to use it in a clustered environment requires the following:

  1. Configuring a data source to access the database that holds the job repository
  2. Targeting the preceding data source to the cluster
  3. Configuring each batch runtime instance to use that data source

  For details about configuring the job repository, and configuring the data source that is used by both the job repository and a batch runtime instance, see Configuring the Batch Runtime to Use a Dedicated Database in *Administering Server Environments for Oracle WebLogic Server*.

- State replication is not available in the batch runtime, so conventional failover is not available. Consequently, a failed batch job cannot be resumed on a backup copy that is running on a different Managed Server instance. However, if all batch runtime instances in a cluster point to the same job repository, and the data source for the database that contains the job repository is also targeted to the cluster. Any batch job that fails on a Managed Server instance can be queried and restarted on any other Managed Server instance in that cluster.

What Types of Objects Cannot Be Clustered?

Certain APIs and internal services cannot be clustered in WebLogic Server.

- File services including file shares

You can still use these services on individual WebLogic Server instances in a cluster. However, the services do not make use of load balancing or failover features.

2

Communications in a Cluster

This chapter introduces how Oracle WebLogic Server clustering implement two key features: load balancing and failover.
The following sections provide information that helps architects and administrators to configure a cluster that meets the needs of a particular Web application.

Choosing WebLogic Server Cluster Messaging Protocols

WebLogic Server supports two cluster messaging protocols: multicast and unicast.

- Multicast: This protocol relies on User Datagram Protocol (UDP) multicast and has been supported in WebLogic Server clusters since WebLogic Server 4.0.
- Unicast: This protocol relies on point-to-point TCP/IP sockets and was added in WebLogic Server 10.0.

Learn about cluster messaging protocols in the following sections:

Using IP Multicast

Multicast is a simple broadcast technology that enables multiple applications to "subscribe" to a given IP address and port number and listen for messages.

> **Note**
>
> A multicast address is an IP address in the range from 224.0.0.0 to 239.255.255.255. The default multicast value used by WebLogic Server is 239.192.0.0. You should not use any multicast address within the range x.0.0.1. Multicast ports have the normal UDP port ranges (0 to 65535). However certain UDP ports are reserved for specific purposes and should generally be avoided.

Multicast broadcasts messages to applications, but it does not guarantee that messages are actually received. If an application's local multicast buffer is full, new multicast messages cannot be written to the buffer and the application is not notified when messages are "dropped." Because of this limitation, WebLogic Server instances allow for the possibility that they may occasionally miss messages that were broadcast over multicast.

The WebLogic Server multicast implementation uses standard UDP multicast to broadcast the cluster messages to a group that is explicitly listening on the multicast address and port over which the message is sent. Since UDP is not a reliable protocol, WebLogic Server builds its own reliable messaging protocol into the messages it sends to detect and retransmit lost messages.

Most operating systems and switches support UDP multicast by default between machines in the same subnet. However, most routers do not support the propagation of UDP multicast messages between subnets by default. In environments that do support UDP multicast message propagation, UDP multicast has a time-to-live (TTL) mechanism built into the protocol. Each time the message reaches a router, the TTL is decremented by 1 before it

routes the message. When the TTL reaches zero, the message will no longer be propagated between networks, making it an effective control for the range of a UDP multicast message. By default, WebLogic Server sets the TTL for its multicast cluster messages to 1, which restricts the message to the current subnet.

When using multicast, the cluster heartbeat mechanism will remove a server instance from the cluster if it misses three heartbeat messages in a row to account for the fact that UDP is not considered a reliable protocol. Since the default heartbeat frequency is one heartbeat every 10 seconds, this means it can take up to 30 seconds to detect that a server instance has left the cluster. Socket death detection or failed connection attempts can also accelerate this detection.

In summary, WebLogic Server multicast cluster messaging protocol:

- Uses a very efficient and scalable peer-to-peer model where a server instance sends each message directly to the network once and the network makes sure that each cluster member receives the message directly from the network.
- Works out of the box in most environments where the cluster members are in a single subnet.
- Requires additional configuration in the router and WebLogic Server (for example multicast TTL) if the cluster members span more than one subnet.
- Uses three consecutive missed heartbeats to remove a server instance from another server's cluster membership list.

To test an environment for its ability to support the WebLogic Server multicast messaging protocol, WebLogic Server provides a Java command-line utility known as MulticastTest.

WebLogic Server uses multicast for all one-to-many communications among server instances in a cluster. This communication includes:

- Cluster-wide JNDI updates: Each WebLogic Server instance in a cluster uses multicast to announce the availability of clustered objects that are deployed or removed locally. Each server instance in the cluster monitors these announcements and updates its local JNDI tree to reflect current deployments of clustered objects. See Cluster-Wide JNDI Naming Service for more details.
- Cluster heartbeats: Each WebLogic Server instance in a cluster uses multicast to broadcast regular "heartbeat" messages that advertise its availability. By monitoring heartbeat messages, server instances in a cluster determine when a server instance has failed. (Clustered server instances also monitor IP sockets as a more immediate method of determining when a server instance has failed.)
- Clusters with many nodes: Multicast communication is the option of choice for clusters with many nodes.

Multicast and Cluster Configuration

Because multicast communications control critical functions related to detecting failures and maintaining the cluster-wide JNDI tree (described in Cluster-Wide JNDI Naming Service) it is important that neither the cluster configuration nor the network topology interfere with multicast communications. The sections that follow provide guidelines for avoiding problems with multicast communication in a cluster.

If Your Cluster Spans Multiple Subnets In a WAN

In many deployments, clustered server instances reside within a single subnet, ensuring multicast messages are reliably transmitted. However, you may want to distribute a WebLogic

Server cluster across multiple subnets in a Wide Area Network (WAN) to increase redundancy, or to distribute clustered server instances over a larger geographical area.

If you choose to distribute a cluster over a WAN (or across multiple subnets), plan and configure your network topology to ensure that multicast messages are reliably transmitted to all server instances in the cluster. Specifically, your network must meet the following requirements:

- Full support of IP multicast packet propagation. In other words, all routers and other tunneling technologies must be configured to propagate multicast messages to clustered server instances.
- Network latency low enough to ensure that most multicast messages reach their final destination in approximately 10 milliseconds.
- Multicast Time-To-Live (TTL) value for the cluster high enough to ensure that routers do not discard multicast packets before they reach their final destination. See Configure Multicast Time-To-Live (TTL).

> **Note**
>
> Distributing a WebLogic Server cluster over a WAN may require network facilities in addition to the multicast requirements described above. For example, you may want to configure load balancing hardware to ensure that client requests are directed to server instances in the most efficient manner (to avoid unnecessary network hops).

Firewalls Can Break Multicast Communication

Although it may be possible to tunnel multicast traffic through a firewall, this practice is not recommended for WebLogic Server clusters. Treat each WebLogic Server cluster as a logical unit that provides one or more distinct services to clients of a Web application. Do not split this logical unit between different security zones. Furthermore, any technologies that potentially delay or interrupt IP traffic can disrupt a WebLogic Server cluster by generating false failures due to missed heartbeats.

Do Not Share the Cluster Multicast Address with Other Applications

Although multiple WebLogic Server clusters can share a single IP multicast address and port, other applications should not broadcast or subscribe to the multicast address and port used by your cluster or clusters. That is, if the machine or machines that host your cluster also host other applications that use multicast communications, make sure that those applications use a different multicast address and port than the cluster does.

Sharing the cluster multicast address with other applications forces clustered server instances to process unnecessary messages, introducing overhead. Sharing a multicast address may also overload the IP multicast buffer and delay transmission of WebLogic Server heartbeat messages. Such delays can result in a WebLogic Server instance being marked as failed, simply because its heartbeat messages were not received in a timely manner.

For these reasons, assign a dedicated multicast address for use by WebLogic Server clusters, and ensure that the address can support the broadcast traffic of all clusters that use the address.

If Multicast Storms Occur

If server instances in a cluster do not process incoming messages on a timely basis, it results in increased network traffic, including negative acknowledgement (NAK) messages and heartbeat re-transmissions. The repeated transmission of multicast packets on a network is referred to as a *multicast storm*, and can stress the network and attached stations, potentially causing end-stations to hang or fail. Increasing the size of the multicast buffers can improve the rate at which announcements are transmitted and received, and prevent multicast storms. See Configure Multicast Buffer Size.

One-to-Many Communication Using Unicast

The WebLogic Server unicast protocol uses standard TCP/IP sockets to send messages between cluster members. Since all networks and network devices support TCP/IP sockets, unicast simplifies out-of-the-box-cluster configuration. It typically requires no additional configuration, regardless of the network topology between cluster members. Additionally, unicast reduces potential network errors that can occur from multicast address conflicts. WebLogic Server uses unicast as its default cluster protocol.

WebLogic Server Unicast Groups

Since TCP/IP sockets are a point-to-point mechanism, all cluster members receive messages directly. To limit the number of sockets required as a cluster grows, WebLogic Server's unicast implementation uses a group leader mechanism. With this mechanism:

- WebLogic Server divides the server instances in a cluster into a fixed number of groups.
- Each group includes one server instance that also functions as the group leader. If the group leader fails, the group elects another group leader.
- To send and receive cluster messages, each server instance in a group makes a TCP/IP socket connection only to the group leader. The group leader connects to all its group members and all other group leaders in the cluster.
- When a group leader receives a cluster message from a server instance in its group, it retransmits the message to all other members in the group and also to every other group leader in the cluster. The other group leaders then retransmit the message to all their group members. This enables each server instance to receive every message in a cluster without requiring that server to establish a connection to every other server instance in the cluster.

When using unicast, server instances send heartbeats to advertise their availability, similar to multicast. By monitoring heartbeat messages, server instances determine when another server instance fails. However, with unicast, the cluster heartbeat mechanism removes a server instance from the cluster if it misses a single heartbeat message, since TCP/IP is a reliable protocol.

Unicast checks for missed heartbeats every 15 seconds, instead of every 10 seconds as in multicast. This extra five seconds allows sufficient time for the message to travel from the remote group's member to the remote group's leader, then to the local group's leader, and finally to the local group's member. Since the default heartbeat frequency is one heartbeat every 10 seconds, this means it should take no more than 15 seconds to detect if a server instance has left the cluster. Socket death detection or failed connection attempts can also accelerate this detection.

Assigning Server Instances to Groups

> **Note**
>
> The algorithm used to assign server instances to groups has been changed from the algorithm used in WebLogic Server 12.1.2 and prior versions. The new algorithm is described in the following section. It has been optimized to provide more flexible scaling of running clusters, and to better support use cases where Managed Servers are added to WebLogic Server clusters while the clusters are running.

The WebLogic Server unicast implementation internally organizes a cluster's server instances into 10 groups. WebLogic Server assigns server instances to groups and sorts server instances within each group according to a server naming pattern. Since a group contains a dynamic number of server instances, asymmetric or empty groups might exist, depending on the number and names of your clustered server instances.

To assign server instances to groups, WebLogic Server separates each server name into two parts: a prefix and an integer. For example, a server instance named `server1` separates into the prefix `<server>` and the integer `<1>`.

You can use any name for server instances. For configured servers, if the server name does not end with an integer, WebLogic Server calculates and assigns an initial value to the server instance. It then uses this value to determine the appropriate group to which it automatically assigns the server instance. For example, server instances `serverA` and `serverB` do not have integers in their names. WebLogic Server uses the entire names for the prefixes and calculates values to use for the integers, such as `728` for `serverA` and `729` for `serverB`.

Dynamic servers always follow this pattern, as a dynamic cluster uses its server template settings to automatically name dynamic servers using a prefix and a sequential integer number.

After associating an integer with each server name, WebLogic Server uses an algorithm to assign server instances to groups based on that integer. Within each group, server instances are first sorted alphabetically by prefix and then sorted by integer.

The first server instance in each group acts as the group leader. Under this allocation model, all server instances in the cluster, whether existing running servers or newly added servers, share a consistent view on group membership and group leader roles.

The following tables demonstrate the unicast naming pattern and how WebLogic Server assigns and sorts server instances into groups. This example uses 10 groups; the cluster contains 15 server instances named `server1` through `server15` and five additional server instances named `serverA` through `serverE`.

Table 2-1 Separating Server Names into Prefixes and Integers

Server Name	Prefix	Integer
`server1`	server	1
`server2`	server	2
`server3`	server	3
`server4`	server	4

Table 2-1 (Cont.) Separating Server Names into Prefixes and Integers

Server Name	Prefix	Integer
server5	server	5
server6	server	6
server7	server	7
server8	server	8
server9	server	9
server10	server	10
server11	server	11
server12	server	12
server13	server	13
server14	server	14
server15	server	15
serverA	serverA	calculated result is 728
serverB	serverB	calculated result is 729
serverC	serverC	calculated result is 730
serverD	serverD	calculated result is 731
serverE	serverE	calculated result is 732

Table 2-2 Assigning Server Instances to Groups

Group	Server Instances Within Group
group0	server10 (group leader), serverC
group1	server1 (group leader), server11, serverD
group2	server2 (group leader), server12, severE
group3	server3 (group leader), server13
group4	server4 (group leader), server14
group5	server5 (group leader), server15
group6	server6 (group leader)
group7	server7 (group leader)
group8	server8 (group leader), serverA
group9	server9 (group leader), serverB

If you add a new server instance named `server16`, WebLogic Server assigns it to `group6`, after `server6`:

```
group6: server6 (group leader), server 16
```

If you add a new server instance named `server20`, WebLogic Server assigns it to `group0`, after `server10`, but before `serverC`:

```
group0: server10 (group leader), server20, serverC
```

If you add a new server named `clonedServer16`, WebLogic Server assigns it to `group6`, before `server6`, as prefixes are sorted before integers. The group leader then changes to `clonedServer16`, as `clonedServer16` is now the first server instance in the group:

```
group6: clonedServer16 (new group leader), server6, server16
```

Configuring Unicast

Configure unicast using `ClusterMBean.setClusterMessagingMode` MBean attribute. The default value of this parameter is `unicast`. Changes made to this MBean are not dynamic. You must restart your cluster for changes to take effect.

To define a specific unicast channel, first you must define a custom network channel for unicast communications with either the `cluster-broadcast` or the `cluster-broadcast-secure` protocol. After defining this custom network channel, you can associate this channel with the cluster by specifying the channel name in the `ClusterMBean.ClusterBroadcastChannel` MBean attribute. When unicast is enabled, servers attempt to use the value defined in this MBean attribute for communications between clusters. If the unicast channel is not explicitly defined, the default network channel is used.

> **Note**
>
> The `ClusterMBean.ClusterBroadcastChannel` attribute is only supported for use only with unicast.

When you configure a WebLogic Server cluster to use unicast communications, make sure to set the listen port and the listen address for all the servers in the cluster. This is necessary so that TCP connections can be successfully established. Specify these ports for both static and dynamic servers, and note that you can use the IP address or the DNS name to set the listen addresses. Alternatively, you can configure a custom Unicast broadcast channel. See Configure Custom Network Channels in the *Oracle WebLogic Remote Console Online Help*.

Considerations When Using Unicast

The following considerations apply when using unicast to handle cluster communications in WebLogic Server 15.1.1.0.0:

- All members of a cluster must use the same message type. Using both multicast and unicast messaging is not allowed.
- Individual cluster members cannot override the cluster messaging type.
- Each server instance must have a unique combination of listen port and listen address.
- The entire cluster must be shut down and restarted to change message modes.
- JMS topics configured for multicasting can access WebLogic clusters configured for unicast because a JMS topic publishes messages on its own multicast address that is independent of the cluster address. However, the following considerations apply:
  - The router hardware configurations that allow unicast clusters may not allow JMS multicast subscribers to work.
  - JMS multicast subscribers need to be in a network hardware configuration that allows multicast accessibility.

    See Using Multicasting with WebLogic JMS in *Developing JMS Applications for Oracle WebLogic Server*.

Considerations for Choosing Unicast or Multicast

Unicast is the default protocol because it simplifies out of the box cluster configuration and because it is likely to meet the majority of user requirements. However, Oracle fully supports both protocols equally. Both protocols require that the cluster members get sufficient processing time to send and receive cluster messages in a timely fashion. This prevents unnecessary cluster membership changes and the inherent resynchronization costs associated with leaving and rejoining the cluster. It is recommended that you eliminate unnecessary cluster membership changes due to over-utilization of available resources.

When using unicast in particular, make sure that the group leaders are not resource constrained since they act as the message relay to deliver a cluster message to the rest of the cluster. Any slowness on their part can impact multiple cluster members and even result in the group electing a new group leader.

Contrast this with multicast, where a slow member can only really impact its own membership to the cluster. Multicast clusters are generally more efficient in terms of cluster message propagation, and therefore tend to be more resilient to oversubscription of resources. For these reasons, multicast may be a better option for very large clusters with high throughput requirements, provided the network environment supports WebLogic Server cluster UDP requirements.

Each protocol has its own benefits.Table 2-3 highlights some of the differences between multicast and unicast.

Table 2-3 Summary of Differences Between Multicast and Unicast

Multicast	Unicast
Uses UDP multicast	Uses TCP/IP
Requires additional configuration to routers, TTL when clustering across multiple subnets	Requires no additional configuration to account for network topology
Requires configuring the multicast listen address and listen port. May need to specify the network interface to use on machines with multiple NICs	Only requires specifying the listen address. Supports using the default channel or a custom network channel for cluster communications
Each message delivered directly to and received directly from the network	Each message is delivered to a group leader, which retransmits the message to other group members (N - 1) and any other group leaders (M - 1), if they exist. The other group leaders then retransmit the message to their group members resulting in up to NxM network messages for every cluster message. Message delivery to each cluster member takes between one and three network hops.
Every server sees every other server	Group leaders act as a message relay point to retransmit messages to its group members and other group leaders
Cluster membership changes require three consecutive missed heartbeat messages to remove a member from the cluster list	Cluster membership changes require only a single missed heartbeat message to remove a member from the cluster

Peer-to-Peer Communication Using IP Sockets

IP sockets provide a simple, high-performance mechanism for transferring messages and data between two applications.

Clustered WebLogic Server instances use IP sockets for:

- Accessing non-clustered objects deployed to another clustered server instance on a different machine.
- Replicating HTTP session states and stateful session EJB states between a primary and secondary server instance.
- Accessing clustered objects that reside on a remote server instance. (This generally occurs only in a multitier cluster architecture, such as the one described in Recommended Multitier Architecture.)

> **Note**
>
> The use of IP sockets in WebLogic Server extends beyond the cluster scenario—all RMI communication takes place using sockets, for example, when a remote Java client application accesses a remote object.

Proper socket configuration is crucial to the performance of a WebLogic Server cluster. For more information, see Tuning Muxers in *Tuning Performance of Oracle WebLogic Server*.

Client Communication by Using Sockets

Clients of a cluster use the Java implementation of socket reader threads. WebLogic Server allows you to configure server affinity load balancing algorithms that reduce the number of IP sockets opened by a Java client application.

A client accessing multiple objects on a server instance will use a single socket. If an object fails, the client will failover to a server instance to which it already has an open socket, if possible. In older version of WebLogic Server, under some circumstances, a client might open a socket to each server instance in a cluster.

For optimal performance, configure enough socket reader threads in the Java Virtual Machine (JVM) that runs the client. For instructions, see Set the Number of Reader Threads on Client Machines.

Cluster-Wide JNDI Naming Service

Clients of a non-clustered WebLogic Server instance access objects and services by using a JNDI-compliant naming service.

The JNDI naming service contains a list of the public services that the server instance offers, organized in a tree structure. A WebLogic Server instance offers a new service by binding into the JNDI tree a name that represents the service. Clients obtain the service by connecting to the server instance and looking up the bound name of the service.

Server instances in a cluster utilize a cluster-wide JNDI tree. A cluster-wide JNDI tree is similar to a single server instance JNDI tree, insofar as the tree contains a list of available services. In addition to storing the names of local services, however, the cluster-wide JNDI tree stores the

services offered by clustered objects (EJBs and RMI classes) from other server instances in the cluster.

Each WebLogic Server instance in a cluster creates and maintains a local copy of the logical cluster-wide JNDI tree. The following sections describe how the cluster-wide JNDI tree is maintained, and how to avoid naming conflicts that can occur in a clustered environment.

> **Note**
>
> Do not use the cluster-wide JNDI tree as a persistence or caching mechanism for application data. Although WebLogic Server replicates a clustered server instance's JNDI entries to other server instances in the cluster, those entries are removed from the cluster if the original instance fails. Also, storing large objects within the JNDI tree can overload multicast or unicast traffic and interfere with the normal operation of a cluster.

How WebLogic Server Creates the Cluster-Wide JNDI Tree

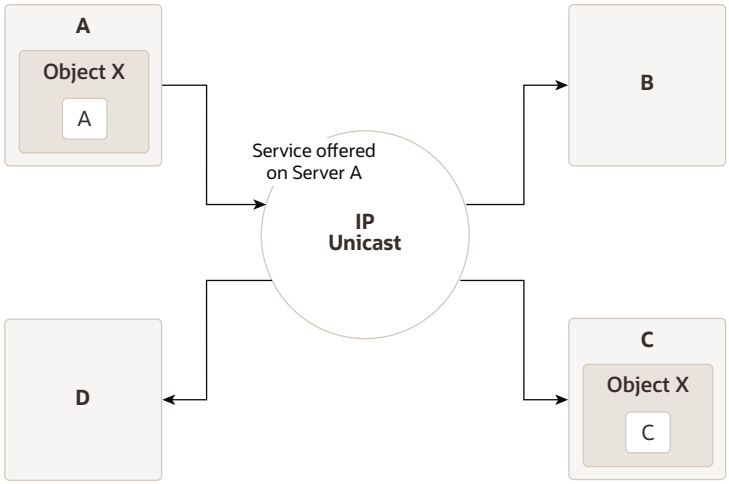
Each WebLogic Server in a cluster builds and maintains its own local copy of the cluster-wide JNDI tree, which lists the services offered by all members of the cluster. Creation of a cluster-wide JNDI tree begins with the local JNDI tree bindings of each server instance. As a server instance boots (or as new services are dynamically deployed to a running server instance), the server instance first binds the implementations of those services to the local JNDI tree. The implementation is bound into the JNDI tree only if no other service of the same name exists.

> **Note**
>
> When you start a Managed Server in a cluster, the server instance identifies other running server instances in the cluster by listening for heartbeats, after a warm-up period specified by the `MemberWarmupTimeoutSeconds` parameter in `ClusterMBean`. The default warm-up period is 30 seconds.

Once the server instance successfully binds a service into the local JNDI tree, additional steps are performed for clustered objects that use replica-aware stubs. After binding the clustered object's implementation into the local JNDI tree, the server instance sends the object's stub to other members of the cluster. Other members of the cluster monitor the multicast or unicast address to detect when remote server instances offer new services.

Figure 2-1 shows a snapshot of the JNDI binding process.

Figure 2-1 Server A Binds an Object in its JNDI Tree, then Unicasts Object Availability



In the previous figure, Server A has successfully bound an implementation of clustered Object X into its local JNDI tree. Because Object X is clustered, it offers this service to all other members of the cluster. Server C is still in the process of binding an implementation of Object X.

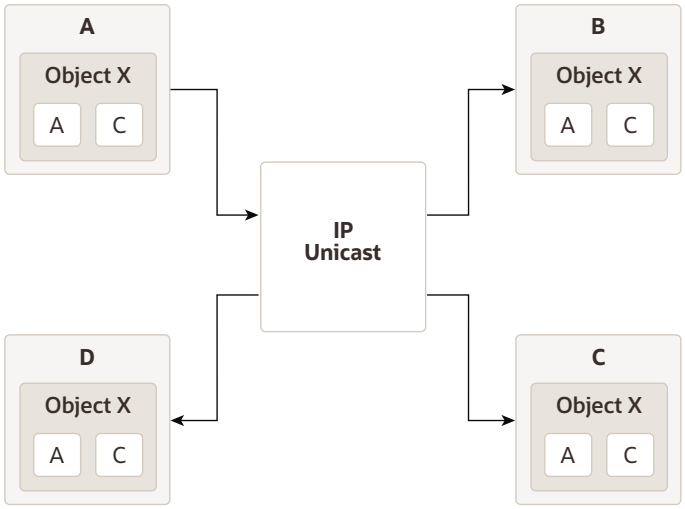
Other server instances in the cluster listening to the multicast or unicast address note that Server A offers a new service for clustered object, X. These server instances update their local JNDI trees to include the new service.

Updating the local JNDI bindings occurs in one of two ways:

- If the clustered service is not yet bound in the local JNDI tree, the server instance binds a new replica-aware stub into the local tree that indicates the availability of Object X on Server A. Servers B and D would update their local JNDI trees in this manner, because the clustered object is not yet deployed on those server instances.
- If the server instance already has a binding for the cluster-aware service, it updates its local JNDI tree to indicate that a replica of the service is also available on Server A. Server C would update its JNDI tree in this manner, because it will already have a binding for the clustered Object X.

In this manner, each server instance in the cluster creates its own copy of a cluster-wide JNDI tree. The same process would be used when Server C announces that Object X has been bound into its local JNDI tree. After all broadcast messages are received, each server instance in the cluster would have identical local JNDI trees that indicate the availability of the object on Servers A and C, as shown in Figure 2-2.

Figure 2-2 Each Server's JNDI Tree is the Same after Unicast Messages are Received



> **Note**
>
> In an actual cluster, Object X would be deployed homogeneously, and an implementation which can invoke the object would be available on all four server instances.

How JNDI Naming Conflicts Occur

Simple JNDI naming conflicts occur when a server instance attempts to bind a non-clustered service that uses the same name as a non-clustered service already bound in the JNDI tree. Cluster-level JNDI conflicts occur when a server instance attempts to bind a clustered object that uses the name of a non-clustered object already bound in the JNDI tree.

WebLogic Server detects simple naming conflicts (of non-clustered services) when those services are bound to the local JNDI tree. Cluster-level JNDI conflicts may occur when new services are advertised over multicast or unicast. For example, if you deploy a pinned RMI object on one server instance in the cluster, you cannot deploy a replica-aware version of the same object on another server instance.

If two server instances in a cluster attempt to bind different clustered objects using the same name, both will succeed in binding the object locally. However, each server instance will refuse to bind the other server instance's replica-aware stub into the JNDI tree due to the JNDI naming conflict. A conflict of this type would remain until one of the two server instances was shut down, or until one of the server instances undeployed the clustered object. This same conflict could also occur if both server instances attempt to deploy a pinned object with the same name.

Deploy Homogeneously to Avoid Cluster-Level JNDI Conflicts

To avoid cluster-level JNDI conflicts, you must homogeneously deploy all replica-aware objects to all WebLogic Server instances in a cluster. Having unbalanced deployments across WebLogic Server instances increases the chance of JNDI naming conflicts during startup or redeployment. It can also lead to unbalanced processing loads in the cluster.

If you must pin specific RMI objects or EJBs to individual server instances, do not replicate the object's bindings across the cluster.

How WebLogic Server Updates the JNDI Tree

When a clustered object is removed (undeployed from a server instance), updates to the JNDI tree are handled similarly to the updates performed when new services are added. The server instance on which the service was undeployed broadcasts a message indicating that it no longer provides the service. Again, other server instances in the cluster that observe the multicast or unicast message update their local copies of the JNDI tree to indicate that the service is no longer available on the server instance that undeployed the object.

Once the client has obtained a replica-aware stub, the server instances in the cluster may continue adding and removing host servers for the clustered objects. As the information in the JNDI tree changes, the client's stub may also be updated. Subsequent RMI requests contain update information as necessary to ensure that the client stub remains up-to-date.

Client Interaction with the Cluster-Wide JNDI Tree

Clients that connect to a WebLogic Server cluster and look up a clustered object obtain a replica-aware stub for the object. This stub contains the list of available server instances that host implementations of the object. The stub also contains the load balancing logic for distributing the load among its host servers.

For more information about replica-aware stubs for EJBs and RMI classes, see Replication and Failover for EJBs and RMIs.

For more information about how WebLogic JNDI is implemented in a clustered environment and how to make your own objects available to JNDI clients, see Using WebLogic JNDI in a Clustered Environment in *Developing JNDI Applications for Oracle WebLogic Server*.

3

Understanding Cluster Configuration

Learn how the information that defines the configuration of a Oracle WebLogic Server cluster is stored and maintained, and the methods you can use to accomplish configuration tasks.

> **Note**
>
> Information in this section pertains to configure a WebLogic domain in which the server instances are not clustered.

This chapter includes the following sections:

Cluster Configuration and config.xml

The `config.xml` file is an XML document that describes the configuration of a WebLogic Server domain.

The `config.xml` file consists of a series of XML elements. The `<domain>` element is the top-level element, and all elements in the domain descend from the `<domain>` element. The `<domain>` element includes child elements, such as the `<server>`, `<cluster>`, `<application>`, and so on. These child elements may have children of their own. For example, the `<server>` element includes the child elements `<WebServer>`, `<SSL>`, and `<Log>`. The `<application>` element includes the child elements `<EJBComponent>` and `<WebAppComponent>`.

Each element has one or more configurable attributes. An attribute defined in `config.dtd` has a corresponding attribute in the configuration API. For example, the Server element has a `ListenPort` attribute, and likewise, the `weblogic.management.configuration.ServerMBean` has a `ListenPort` attribute. Configurable attributes are readable and writable, that is, `ServerMBean` has a `getListenPort` and a `setListenPort` method.

To learn more about the `config.xml` file, see Domain Configuration Files in *Understanding Domain Configuration for Oracle WebLogic Server*.

Role of the Administration Server

The Administration Server is the WebLogic Server instance that configures and manages the WebLogic Server instances in its domain.

A domain can include multiple WebLogic Server clusters and non-clustered WebLogic Server instances. Strictly speaking, a domain could consist of only one WebLogic Server instance. However, in that case that sole server instance would be an Administration Server, because each domain must have exactly one Administration Server.

There are varieties of ways to invoke the services of the Administration Server to accomplish configuration tasks, as described in Methods of Configuring Clusters. Whichever method you use, the Administration Server for a cluster must be running when you modify the configuration.

When the Administration Server starts, it loads the `config.xml` for the domain. It looks for `config.xml` in the directory:

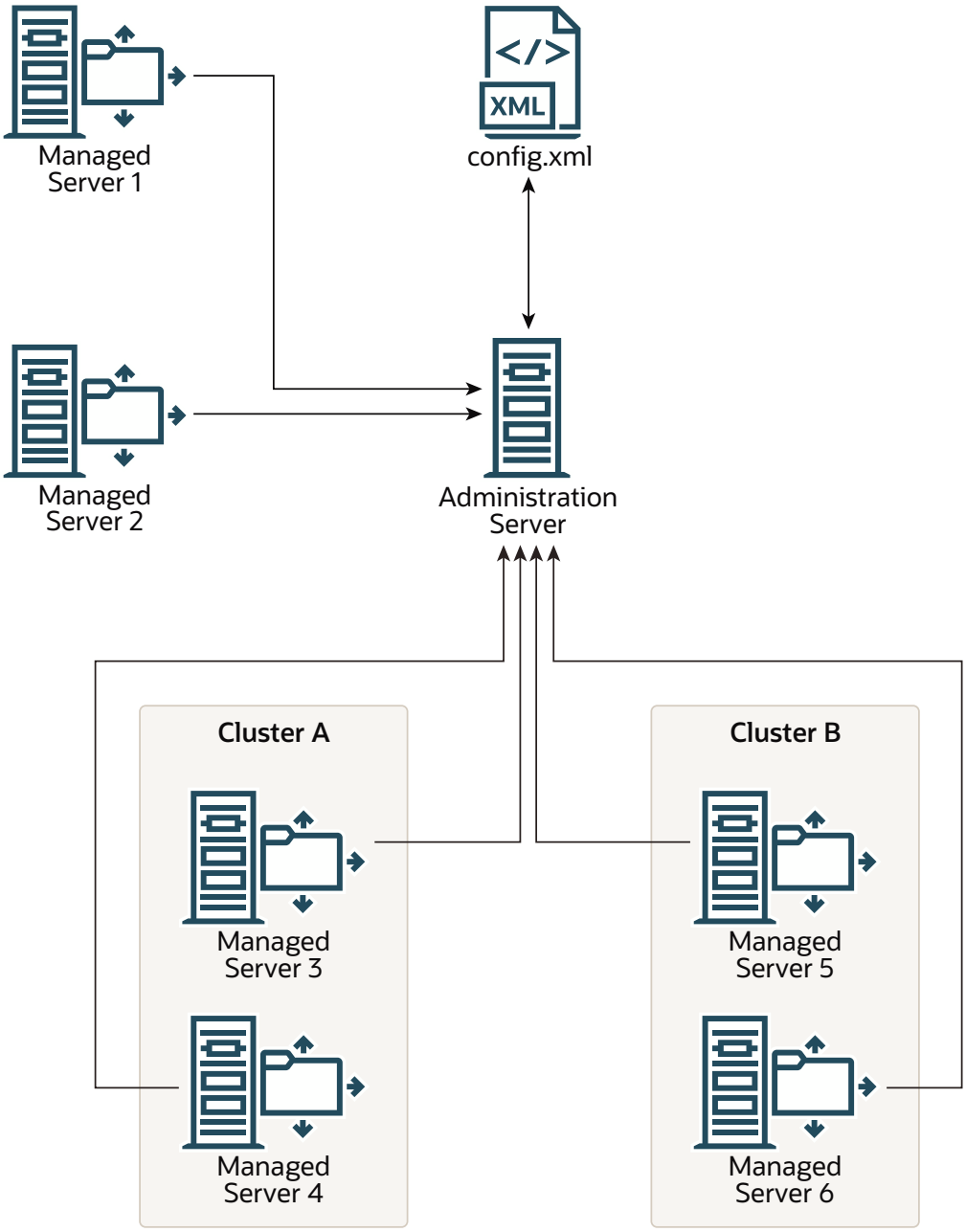
ORACLE_HOME`/user_projects/domains/`*domain_name*`/config`

where *domain_name* is a domain-specific directory, with the same name as the domain.

> **Note**
>
> Do not add non-configuration files in the `config` directory or subdirectories. Non-configuration files include log (.log) and lock (.lck) files. Administration Server replicates the `config` directory in all Managed Server instances. Storing non-configuration files in the `config` directory can cause performance issues in the domain.

Figure 3-1 shows a typical production environment that contains an Administration Server and multiple WebLogic Servers instances. When you start the server instances in such a domain, the Administration Server is started first. As each additional server instance is started, it contacts the Administration Server for its configuration information. In this way, the Administration Server operates as the central control entity for the configuration of the entire domain.

Figure 3-1 WebLogic Server Configuration



What Happens if the Administration Server Fails?

The failure of an Administration Server for a domain does not affect the operation of Managed Servers in the domain. If an Administration Server for a domain becomes unavailable while the server instances it manages (clustered or otherwise) are up and running, those Managed Servers continue to run. If the domain contains clustered server instances, the load balancing and failover capabilities supported by the domain configuration remain available, even if the Administration Server fails.

> **Note**
>
> If an Administration Server fails because of a hardware or software failure on its host machine, other server instances on the same machine may be similarly affected. However, the failure of an Administration Server itself does not interrupt the operation of Managed Servers in the domain.

For instructions on re-starting an Administration Server, see Avoiding and Recovering from Server Failure in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

How Dynamic Configuration Works

WebLogic Server allows you to change the configuration attributes of domain resources dynamically—while server instances are running. In most cases you do not need to restart the server instance for your changes to take effect. When an attribute is reconfigured, the new value is immediately reflected in both the current runtime value of the attribute and the persistent value stored in `config.xml`.

However, not all configuration changes are applied dynamically. For example, if you change a Managed Server's `ListenPort` value, the new port will not be used until the next time you start the Managed Server. The updated value is stored in `config.xml`, but the runtime value is not affected.

The WebLogic Remote Console validates attribute changes, checking for out-of-range errors and data type mismatch errors, and displays an error message for erroneous entries.

After the WebLogic Remote Console is connected to an Administration Server, if another process captures the listen port assigned to the Administration Server, you should stop the process that captured the port. If you are not able to remove the process that captured the listen port, edit the `config.xml` file to change the `ListenPort` value.

Application Deployment for Clustered Configurations

Here is a brief introduction to the application deployment process.

For more information about introduction to the application deployment process, see *Deploying Applications to Oracle WebLogic Server*.

For instructions on how to perform common deployment tasks, see Deploy Applications.

Deployment Methods

You can deploy an application to a cluster using the following methods:

- WebLogic Remote Console

  The WebLogic Remote Console is a graphical user interface (GUI) to the Administration Service.

- weblogic.Deployer

  The weblogic.Deployer utility is a Java-based deployment tool that provides a command-line interface to the WebLogic Server deployment API.

- WebLogic Scripting Tool (WLST)

The WLST is a command-line interface that you can use to automate domain configuration tasks, including application deployment configuration and deployment operations.

These deployment tools are discussed in Deployment Tools in *Deploying Applications to Oracle WebLogic Server*.

Regardless of the deployment tool you use, when you initiate the deployment process, specify the components to be deployed and the targets to which they will be deployed—your cluster, or individual server instances within the cluster or domain.

The Administration Server for the domain manages the deployment process, communicating with the Managed Servers in the cluster throughout the process. Each Managed Server downloads components to be deployed, and initiates local deployment tasks. The deployment state is maintained in the relevant MBeans for the component being deployed. See Deployment Management API.

> **Note**
>
> You must package applications before you deploy them to WebLogic Server. See Packaging Applications Using wlpackage in *Developing Applications for Oracle WebLogic Server*.

Introduction to Two-Phase Deployment

In WebLogic Server, applications are deployed in two phases. Before starting, WebLogic Server determines the availability of the Managed Servers in the cluster.

First Phase of Deployment

During the first phase of deployment, application components are distributed to the target server instances, and the planned deployment is validated to ensure that the application components can be successfully deployed. During this phase, user requests to the application being deployed are not allowed.

Failures encountered during the distribution and validation process will result in aborted deployment on all server instances (including those upon which the validation succeeded). Files that have been staged will not be removed; however, container-side changes performed during the preparation will be reverted.

Second Phase of Deployment

After the application components have been distributed to targets and validated, they are fully deployed on the target server instances, and the deployed application is made available to clients.

When a failure is encountered during the second phase of deployment, the server starts with one of the following behaviors:

- If a failure occurs while deploying to the target server instances, the server instance will start in ADMIN state. See ADMIN State in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.
- If cluster member fails to deploy an application, the application that failed to deploy is made unavailable.

Guidelines for Deploying to a Cluster

Ideally, all Managed Servers in a cluster should be running and available during the deployment process. Deploying applications while some members of the cluster are unavailable is not recommended. Before deploying applications to a cluster, ensure that all Managed Servers in the cluster are running and reachable by the Administration Server.

> **Note**
>
> If you deploy an application to a Managed Server that is partitioned at the time of deployment—running but not reachable by the Administration Server—problems accessing the Managed Server can occur when that Managed Server rejoins the cluster. During the synchronization period, while other clustered Managed Servers re-establish communications with the previously partitioned server instance, user requests to the deployed applications and attempts to create secondary sessions on that server instance will fail. The risk of this circumstance occurring can be reduced by setting `ClusterConstraintsEnabled`, as described in Enforcing Consistent Deployment to All Configured Cluster Members in *Deploying Applications to Oracle WebLogic Server*.

Cluster membership should not change during the deployment process. After initiating deployment, do not:

- Add or remove Managed Servers to the target cluster.
- Shut down Managed Servers in the target cluster.

WebLogic Server Supports "Relaxed Deployment" Rules

Previous versions of WebLogic Server imposed these restrictions on deployment to clusters:

- No partial deployment: If one or more of the Managed Servers in the cluster are unavailable, the deployment process is terminated, and an error message is generated, indicating that unreachable Managed Servers should be either restarted or removed from the cluster before attempting deployment.
- Pinned services cannot be deployed to multiple Managed Servers in a cluster: If an application is not deployed to the cluster, you can deploy it to only one Managed Server in the cluster.

See the following sections:

Deployment to a Partial Cluster is Allowed

By default, Oracle WebLogic Server allows deployment to a partial cluster. If one or more of the Managed Servers in the cluster are unavailable, the following message may be displayed:

```
Unable to contact "servername". Deployment is deferred until "servername"
becomes available.
```

When the unreachable Managed Server becomes available, deployment to that server instance will be initiated. Until the deployment process is completed, the Managed Server may experience failures related to missing or out-of-date classes.

Deploying to Complete Clusters in WebLogic Server

You can ensure that deployment is only performed if all Managed Servers in the cluster are reachable by setting `ClusterConstraintsEnabled`. When `ClusterConstraintsEnabled` is set to "true", a deployment to a cluster succeeds only if all members of the cluster are reachable and all can deploy the specified files. See Enforcing Consistent Deployment to All Configured Cluster Members in *Deploying Applications to Oracle WebLogic Server*.

Pinned Services can be Deployed to Multiple Managed Servers.

It is possible to target a pinned service to multiple Managed Servers in a cluster. This practice is not recommended. The load-balancing capabilities and scalability of your cluster can be negatively affected by deploying a pinned service to multiple Managed Servers in a cluster. If you target a pinned service to multiple Managed Servers, the following message is printed to the server logs:

```
Adding server servername of cluster clustername as a target for
module modulename. This module also includes server servername that
belongs to this cluster as one of its other targets. Having multiple
individual servers in a cluster as targets instead of having the entire
cluster as the target can result in non-optimal load balancing and
scalability. Hence this is not usually recommended.
```

Methods of Configuring Clusters

You can configure clusters using any one of several WebLogic Server administration tools and methods, such as the Configuration Wizard, WebLogic Remote Console, FMWC, WebLogic Server API, WLST, and Java Management Extensions (JMX).

- Configuration Wizard

  The Configuration Wizard is the recommended tool for creating a new domain or cluster. See Introduction in *Creating WebLogic Domains Using the Configuration Wizard*.

  For more information about creating and configuring a cluster, see Clusters in *Creating WebLogic Domains Using the Configuration Wizard*.

- WebLogic Remote Console and FMWC

  The WebLogic Remote Console and FMWC are GUIs to the Administration Service. They allow you to perform a variety of domain configuration and monitoring functions.

- WebLogic Server API

  You can write a program to modify the configuration attributes, based on the configuration API provided with WebLogic Server. This method is not recommended for initial cluster implementation.

- WebLogic Scripting Tool (WLST)

  The WLST is a command-line scripting interface that system administrators and operators use to monitor and manage WebLogic Server instances and domains. See *Understanding the WebLogic Scripting Tool*.

- Java Management Extensions (JMX)

  JMX is a solution for monitoring and managing resources on a network. WebLogic Server provides a set of MBeans that you can use to configure, monitor, and manage WebLogic Server resources through JMX.

4

Load Balancing in a Cluster

Oracle WebLogic Server clusters provide load balancing support for different types of objects. Learn the related planning and configuration considerations for architects and administrators.

For information about load balancing with the ReadyApp framework, see Using the ReadyApp Framework in *Deploying Applications to Oracle WebLogic Server*.

This chapter includes the following sections:

Load Balancing for Servlets and JSPs

You can accomplish load balancing of servlets and JSPs with the built-in load balancing capabilities of a WebLogic proxy plug-in, with separate load balancing hardware, or NGINX.

> **Note**
>
> In addition to distributing HTTP traffic, external load balancers can distribute initial context requests that come from Java clients over `t3` and the default channel. For a discussion of object-level load balancing in WebLogic Server, see Load Balancing for EJBs and RMI Objects .

Load Balancing with a Proxy Plug-in

The WebLogic proxy plug-in maintains a list of WebLogic Server instances that host a clustered servlet or JSP, and forwards HTTP requests to those instances on a round-robin basis. For more information about this load balancing method, see Round-Robin Load Balancing.

The plug-in also provides the logic necessary to locate the replica of a client's HTTP session state if a WebLogic Server instance should fail.

WebLogic Server supports the following Web servers and associated proxy plug-ins:

- WebLogic Server with the `HttpClusterServlet`
- Oracle HTTP Server with the Oracle HTTP Server (proxy) plug-in
- Apache with the Apache Server (proxy) plug-in
- Microsoft Internet Information Server with the Microsoft-IIS (proxy) plug-in

For instructions on setting up proxy plug-ins, see Configure Proxy Plug-Ins.

How Session Connection and Failover Work with a Proxy Plug-in

For a description of connection and failover for HTTP sessions in a cluster with proxy plug-ins, see Accessing Clustered Servlets and JSPs Using a Proxy.

Load Balancing HTTP Sessions with an External Load Balancer

Clusters that employ a hardware load balancing solution can use any load balancing algorithm supported by the hardware. These can include advanced load-based balancing strategies that monitor the utilization of individual machines.

Load Balancer Configuration Requirements

If you choose to use load balancing hardware instead of a proxy plug-in, it must support a compatible passive or active cookie persistence mechanism, and SSL persistence.

- Passive Cookie Persistence

  Passive cookie persistence enables WebLogic Server to write a cookie containing session parameter information through the load balancer to the client. For information about the session cookie and how a load balancer uses session parameter data to maintain the relationship between the client and the primary WebLogic Server hosting a HTTP session state, see Load Balancers and the WebLogic Session Cookie.

- Active Cookie Persistence

  You can use certain active cookie persistence mechanisms with WebLogic Server clusters, provided the load balancer does not modify the WebLogic Server cookie. WebLogic Server clusters do not support active cookie persistence mechanisms that overwrite or modify the WebLogic HTTP session cookie. If the load balancer's active cookie persistence mechanism works by adding its own cookie to the client session, no additional configuration is required to use the load balancer with a WebLogic Server cluster.

- SSL Persistence

  When SSL persistence is used, the load balancer performs all encryption and decryption of data between clients and the WebLogic Server cluster. The load balancer then uses the plain text cookie that WebLogic Server inserts on the client to maintain an association between the client and a particular server in the cluster.

Load Balancers and the WebLogic Session Cookie

A load balancer that uses passive cookie persistence can use a string in the WebLogic session cookie to associate a client with the server hosting its primary HTTP session state. The string uniquely identifies a server instance in the cluster. You must configure the load balancer with the offset and length of the string constant. The correct values for the offset and length depend on the format of the session cookie.

The format of a session cookie is:

```
sessionid!primary_server_id!secondary_server_id
```

where:

- `sessionid` is a randomly generated identifier of the HTTP session. The length of the value is configured by the `IDLength` parameter in the `<session-descriptor>` element in the `weblogic.xml` file for an application. By default, the `sessionid` length is 52 bytes.
- `primary_server_id` and `secondary_server_id` are 10 character identifiers of the primary and secondary hosts for the session.

> **Note**
>
> For sessions using non-replicated memory, cookie, or file-based session persistence, the `secondary_server_id` is not present. For sessions that use in-memory replication, if the secondary session does not exist, the `secondary_server_id` is "NONE".

For general instructions on configuring load balancers, see Configuring Load Balancers that Support Passive Cookie Persistence. For instructions on configuring BIG-IP, see Configuring BIG-IP Hardware with Clusters.

Related Programming Considerations

For programming constraints and recommendations for clustered servlets and JSPs, see Programming Considerations for Clustered Servlets and JSPs.

How Session Connection and Failover Works with a Load Balancer

For a description of connection and failover for HTTP sessions in a cluster with load balancing hardware, see Accessing Clustered Servlets and JSPs with Load Balancing Hardware.

Load Balancing for EJBs and RMI Objects

Learn the WebLogic Server load balancing algorithms for EJBs and RMI objects. The load balancing algorithm for an object is maintained in the replica-aware stub obtained for a clustered object.

By default, WebLogic Server clusters use round-robin load balancing, described in Round-Robin Load Balancing. You can configure a different default load balancing method for a cluster by using the WebLogic Remote Console to set `weblogic.cluster.defaultLoadAlgorithm`. For instructions, see Configure Load Balancing Method for EJBs and RMIs. You can also specify the load balancing algorithm for a specific RMI object using the `-loadAlgorithm` option in `rmic`, or with the `home-load-algorithm` or `stateless-bean-load-algorithm` in an EJB's deployment descriptor. A load balancing algorithm that you configure for an object overrides the default load balancing algorithm for the cluster.

In addition to the standard load balancing algorithms, WebLogic Server supports custom parameter-based routing. See Parameter-Based Routing for Clustered Objects.

Also, external load balancers can distribute initial context requests that come from Java clients over `t3` and the default channel. However, because WebLogic Server load balancing for EJBs and RMI objects is controlled using replica-aware stubs, including situations where server affinity is employed, you should not route client requests, following the initial context request, through the load balancers. When using the `t3` protocol with external load balancers, you must ensure that only the initial context request is routed through the load balancers, and that subsequent requests are routed and controlled using WebLogic Server load balancing.

Oracle advises against using the `t3s` protocol with external load balancers. In cases where the use of `t3` and SSL with external load balancers is required, Oracle recommends using `t3` tunneling through HTTPS. In cases where server affinity is required, you must use HTTP session IDs for routing requests and terminate SSL at the load balancer, performing session-based routing to enable appropriate routing requests based on session IDs.

> **Note**
>
> Oracle does not recommend enabling tunneling on channels that are available external to the firewall.

Round-Robin Load Balancing

WebLogic Server uses the round-robin algorithm as the default load balancing strategy for clustered object stubs when no algorithm is specified. This algorithm is supported for RMI objects and EJBs. It is also the method used by WebLogic proxy plug-ins.

The round-robin algorithm cycles through a list of WebLogic Server instances in order. For clustered objects, the server list consists of WebLogic Server instances that host the clustered object. For proxy plug-ins, the list consists of all WebLogic Server instances that host the clustered servlet or JSP.

The advantages of the round-robin algorithm are that it is simple, cheap and very predictable. The primary disadvantage is that there is some chance of *convoying*. Convoying occurs when one server is significantly slower than the others. Because replica-aware stubs or proxy plug-ins access the servers in the same order, a slow server can cause requests to "synchronize" on the server, then follow other servers in order for future requests.

> **Note**
>
> WebLogic Server does not always load balance an object's method calls. See Optimization for Collocated Objects.

Weight-Based Load Balancing

This algorithm applies only to EJB and RMI object clustering.

Weight-based load balancing is improved than the round-robin algorithm by taking a pre-assigned weight into account for each server. In the WebLogic Remote Console, go to **Environment**: **Servers**: *myServer*, and click the **Clusters** tab. On the **Clusters** page, in the **Cluster Weight** field, assign each server a numerical weight between 1 and 100. This value determines the proportion of the load the server will withstand relative to other servers. If all servers have the same weight, each of them will withstand an equal proportion of the load. If one server has weight 50 and all other servers have weight 100, then the server with the weight of 50 will withstand half as much as any other server. This algorithm makes it possible to apply the advantages of the round-robin algorithm to clusters that are not homogeneous.

If you use the weight-based algorithm, carefully determine the relative weights to assign to each server instance. Factors to consider include:

- The processing capacity of the server's hardware in relationship to other servers (for example, the number and performance of CPUs dedicated to WebLogic Server).
- The number of non-clustered ("pinned") objects each server hosts.

If you change the specified weight of a server and reboot it, the new weighting information is propagated throughout the cluster via the replica-aware stubs. For related information see Cluster-Wide JNDI Naming Service.

> **Note**
>
> WebLogic Server does not always load balance an object's method calls. See Optimization for Collocated Objects.
>
> In this version of WebLogic Server, weight-based load balancing is not supported for objects that communicate using the RMI/IIOP protocol.

Random Load Balancing

The random method of load balancing applies only to EJB and RMI object clustering.

In random load balancing, requests are routed to servers at random. Random load balancing is recommended only for homogeneous cluster deployments, where each server instance runs on a similarly configured machine. A random allocation of requests does not allow for differences in processing power among the machines upon which server instances run. If a machine hosting servers in a cluster has significantly less processing power than other machines in the cluster, random load balancing will give the less powerful machine as many requests as it gives more powerful machines.

Random load balancing distributes requests evenly across server instances in the cluster, increasingly so as the cumulative number of requests increases. Over a small number of requests the load may not be balanced exactly evenly.

Disadvantages of random load balancing include the slight processing overhead incurred by generating a random number for each request, and the possibility that the load may not be evenly balanced over a small number of requests.

> **Note**
>
> WebLogic Server does not always load balance an object's method calls. See Optimization for Collocated Objects.

Server Affinity Load Balancing Algorithms

WebLogic Server provides three load balancing algorithms for RMI objects that provide *server affinity.* Server affinity turns off load balancing for external client connections; instead, the client considers its existing connections as WebLogic Server instances when choosing the server instance to access an object. If an object is configured for server affinity, the client-side stub attempts to choose a server instance to which it is already connected, and continues to use the same server instance for method calls. All stubs on that client attempt to use that server instance. If the server instance becomes unavailable, the stubs failover, if possible, to a server instance to which the client is already connected.

The purpose of server affinity is to minimize the number of IP sockets opened between external Java clients and server instances in a cluster. WebLogic Server accomplishes this by causing method calls on objects to "stick" to an existing connection, instead of being load balanced among the available server instances. With server affinity algorithms, the less costly server-to-server connections are still load-balanced according to the configured load balancing algorithm.

> **Note**
>
> Load balancing is disabled only for external client connections.

Server affinity is used in combination with one of the standard load balancing methods: Round-Robin, Weight-Based, or Random.

- Round-Robin-affinity: Server affinity governs connections between external Java clients and server instances; round-robin load balancing is used for connections between server instances.
- Weight-Based-affinity: Server affinity governs connections between external Java clients and server instances; weight-based load balancing is used for connections between server instances.
- Random-affinity: Server affinity governs connections between external Java clients and server instances; random load balancing is used for connections between server instances.

For more information about load balancing algorithms that provide server affinity, see Round-Robin Affinity, Weight-Based Affinity, and Random-Affinity.

Server Affinity and Initial Context

A client can request an initial context from a particular server instance in the cluster, or from the cluster by specifying the cluster address in the URL. The connection process varies depending on how the context is obtained.

- If the initial context is requested from a specific Managed Server, the context is obtained using a new connection to the specified server instance.
- If the initial context is requested from a cluster by default, context requests are load balanced on a round-robin basis among the clustered server instances. To reuse an existing connection between a particular JVM and the cluster, set `ENABLE_SERVER_AFFINITY` to `true` in the hash table of `weblogic.jndi.WLContext` properties you specify when obtaining context. (If a connection is not available, a new connection is created.) `ENABLE_SERVER_AFFINITY` is only supported when the context is requested from the cluster address.

Server Affinity and IIOP Client Authentication Using CSIv2

If you use WebLogic Server Common Secure Interoperability (CSIv2) functionality to support stateful interactions with the WebLogic Server Java EE Application Client (thin client), you must use an affinity-based load balancing algorithm to ensure that method calls stick to a server instance. Otherwise, all remote calls will be authenticated. To prevent redundant authentication of stateful CSIv2 clients, use one of the load balancing algorithms described in Round-Robin Affinity, Weight-Based Affinity, and Random-Affinity.

Round-Robin Affinity, Weight-Based Affinity, and Random-Affinity

WebLogic Server has the following three load balancing algorithms that provide server affinity:

- round-robin-affinity
- weight-based-affinity
- random-affinity

Server affinity is supported for all types of RMI objects including JMS objects, all EJB home interfaces, and stateless EJB remote interfaces.

The server affinity algorithms consider existing connections between an external Java client and server instances in balancing the client load among WebLogic Server instances. Server affinity:

- Turns off load balancing between external Java clients and server instances.
- Causes method calls from an external Java client to stick to a server instance to which the client has an open connection, assuming that the connection supports the necessary protocol and QOS.
- In the case of failure, causes the client to failover to a server instance to which it has an open connection, assuming that the connection supports the necessary protocol and QOS.
- Does not affect the load balancing performed for server-to-server connections.

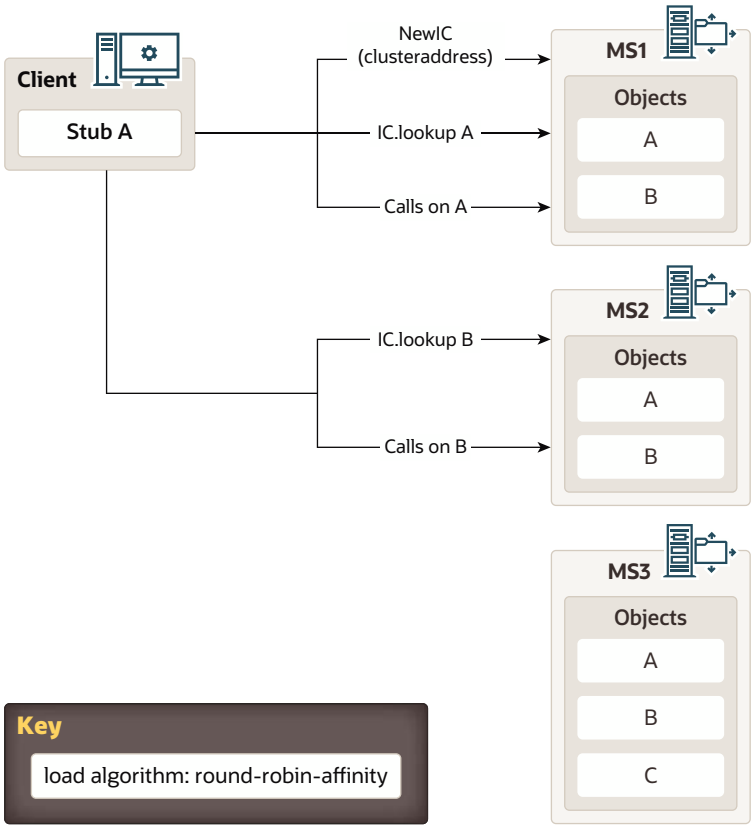
Server Affinity Examples

The following examples illustrate the effect of server affinity under a variety of circumstances. In each example, the objects deployed are configured for round-robin-affinity.

Example 1: Context From Cluster

In the example shown in Figure 4-1, the client obtains context from the cluster. Lookups on the context and object calls stick to a single connection. Requests for new initial context are load balanced on a round-robin basis.

Figure 4-1 Client Obtains Context From the Cluster

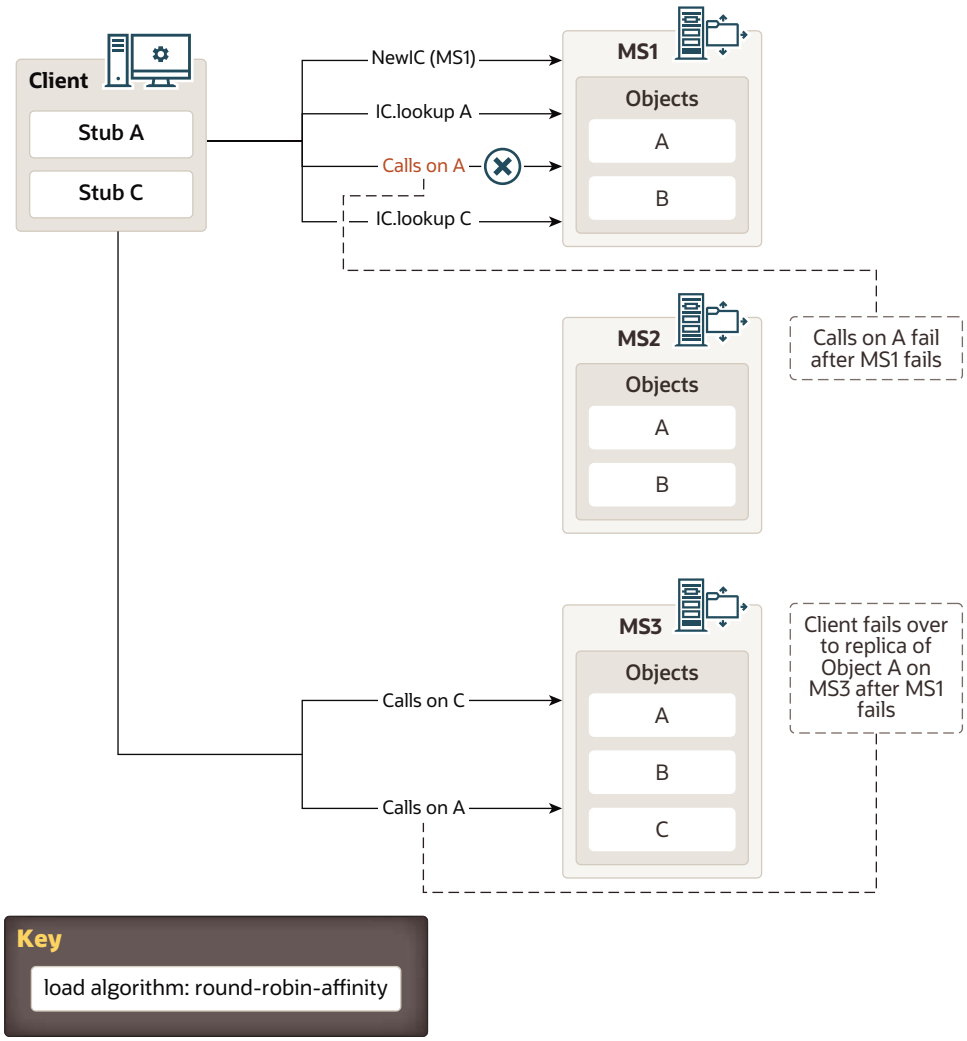


1. Client requests a new initial context from the cluster (`Provider_URL=clusteraddress`) and obtains the context from MS1.
2. Client does a lookup on the context for Object A. The lookup goes to MS1.
3. Client issues a call to Object A. The call goes to MS1, to which the client is already connected. Additional method calls to Object A stick to MS1.
4. Client requests a new initial context from the cluster (`Provider_URL=clusteraddress`) and obtains the context from MS2.
5. Client does a lookup on the context for Object B. The call goes to MS2, to which the client is already connected. Additional method calls to Object B stick to MS2.

Example 2: Server Affinity and Failover

The example shown in Figure 4-2 illustrates the effect that server affinity has on object failover. When a Managed Server goes down, the client fails over to another Managed Server to which it has a connection.

Figure 4-2 Server Affinity and Failover



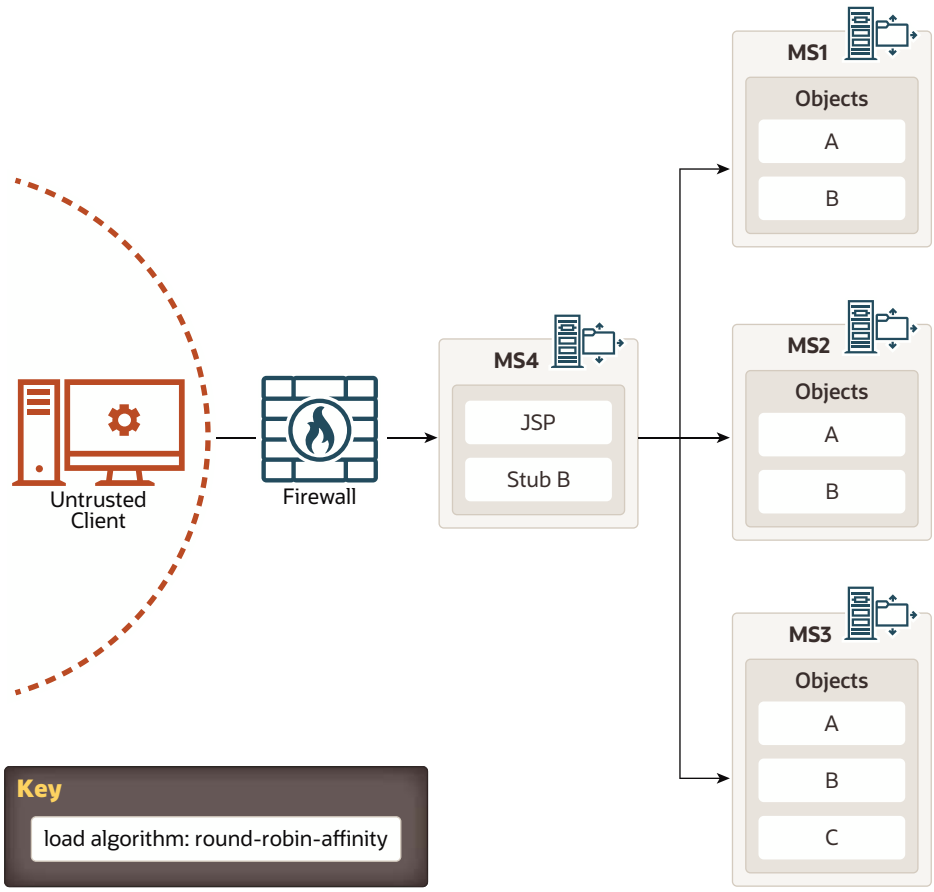
1. Client requests a new initial context from MS1.

2. Client does a lookup on the context for Object A. The lookup goes to MS1.
3. Client makes a call to Object A. The call goes to MS1, to which the client is already connected. Additional calls to Object A stick to MS1.
4. The client obtains a stub for Object C, which is pinned to MS3. The client opens a connection to MS3.
5. MS1 fails.
6. Client makes a call to Object A. The client no longer has a connection to MS1. Because the client is connected to MS3, it fails over to a replica of Object A on MS3.

Example 3: Server affinity and server-to-server connections

The example shown in Figure 4-3 illustrates the fact that server affinity does not affect the connections between server instances.

Figure 4-3 Server Affinity and Server-to-Server Connections



1. A JSP on MS4 obtains a stub for Object B.
2. The JSP selects a replica on MS1. For each method call, the JSP cycles through the Managed Servers upon which Object B is available, on a round-robin basis.

Parameter-Based Routing for Clustered Objects

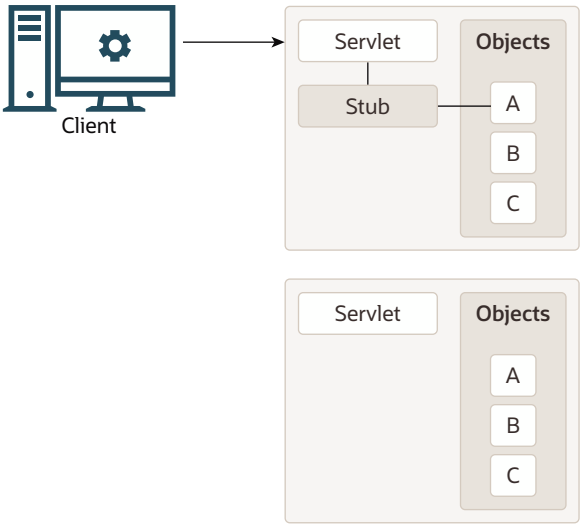
Parameter-based routing allows you to control load balancing behavior at a lower level. Any clustered object can be assigned a `CallRouter`. This is a class that is called before each

invocation with the parameters of the call. The `CallRouter` is free to examine the parameters and return the name server to which the call should be routed. For information about creating custom `CallRouter` classes, see Parameter-Based Routing for Clustered Objects in *Developing RMI Applications for Oracle WebLogic Server*.

Optimization for Collocated Objects

WebLogic Server does not always load balance an object's method calls. In most cases, it is more efficient to use a replica that is collocated with the stub itself, rather than using a replica that resides on a remote server. Figure 4-4 illustrates this.

Figure 4-4 Collocation Optimization Overrides Load Balancer Logic for Method Call



In this example, a client connects to a servlet hosted by the first WebLogic Server instance in the cluster. In response to client activity, the servlet obtains a replica-aware stub for Object A. Because a replica of Object A is also available on the same server instance, the object is said to be collocated with the client's stub.

WebLogic Server always uses the local, collocated copy of Object A, rather than distributing the client's calls to other replicas of Object A in the cluster. It is more efficient to use the local copy, because doing so avoids the network overhead of establishing peer connections to other servers in the cluster.

This optimization is often overlooked when planning WebLogic Server clusters. The collocation optimization is also frequently confusing for administrators or developers who expect or require load balancing on each method call. If your Web application is deployed to a single cluster, the collocation optimization overrides any load balancing logic inherent in the replica-aware stub.

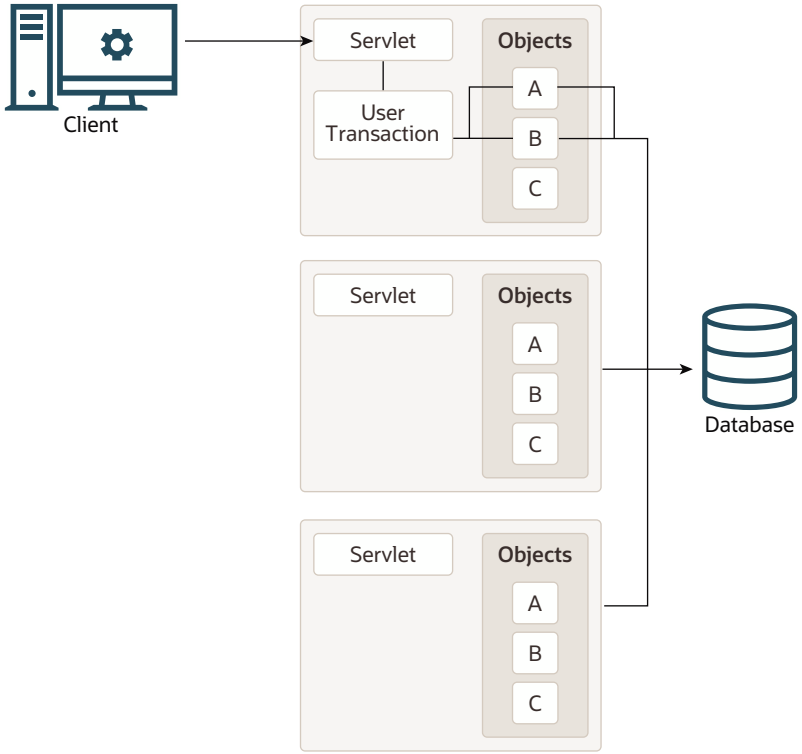
If you require load balancing on each method call to a clustered object, see Recommended Multitier Architecture for information about how to plan your WebLogic Server cluster accordingly.

Transactional Collocation

As an extension to the basic collocation strategy, WebLogic Server attempts to use collocated clustered objects that are enlisted as part of the same transaction. When a client creates a

`UserTransaction` object, WebLogic Server attempts to use object replicas that are collocated with the transaction. This optimization is depicted in the example shown in Figure 4-5.

Figure 4-5 Collocation Optimization Extends to Other Objects in Transaction



In this example, a client attaches to the first WebLogic Server instance in the cluster and obtains a `UserTransaction` object. After beginning a new transaction, the client looks up Objects A and B to do the work of the transaction. In this situation WebLogic Server always attempts to use replicas of A and B that reside on the same server as the `UserTransaction` object, regardless of the load balancing strategies in the stubs for A and B.

This transactional collocation strategy is even more important than the basic optimization described in Optimization for Collocated Objects. If remote replicas of A and B were used, added network overhead would be incurred for the duration of the transaction, because the peer connections for A and B would be locked until the transaction committed. WebLogic Server. By using collocating clustered objects during a transaction, WebLogic Server reduces the network load for accessing the individual objects.

XA Transaction Cluster Affinity

XA transaction cluster affinity allows server instances that are participating in a global transactions to service related requests rather than load-balancing these requests to other member servers. When `Enable Transaction Affinity=true`, cluster throughput is increased by:

- Reducing inter-server transaction coordination traffic.
- Improving resource utilization, such as reducing JDBC connections.
- Simplifying asynchronous processing of transactions.

If the cluster does not have a member participating in the transaction, the request is load balanced to an available cluster member. If the selected cluster member fails, the JTA Transaction Recovery Service can be migrated using the Roadmap for Configuring Automatic Migration of the JTA Transaction Recovery Service.

For more information about configuring the cluster, see Configure Clusters in the *Oracle WebLogic Remote Console Online Help*. You can also enable XA transaction affinity on the command line using `-Dweblogic.cluster.TxnAffinityEnabled=true`.

Load Balancing for JMS

WebLogic Server JMS supports server affinity for distributed JMS destinations and client connections.

By default, a WebLogic Server cluster uses the round-robin method to load balance objects. To use a load balancing algorithm that provides server affinity for JMS objects, you must configure the desired method for the cluster as a whole. You can configure the load balancing algorithm by using the WebLogic Remote Console to set `weblogic.cluster.defaultLoadAlgorithm`. For instructions, see Configure Load Balancing Method for EJBs and RMIs .

Server Affinity for Distributed JMS Destinations

Server affinity is supported for JMS applications that use the distributed destination feature; this feature is not supported for standalone destinations. If you configure server affinity for JMS connection factories, a server instance that is load balancing consumers or producers across multiple members of a distributed destination will first attempt to load balance across any destination members that are also running on the same server instance.

For detailed information on how the JMS connection factory's Server Affinity Enabled option affects the load balancing preferences for distributed destination members, see How Distributed Destination Load Balancing Is Affected When Using the Server Affinity Enabled Attribute in *Administering JMS Resources for Oracle WebLogic Server*.

Initial Context Affinity and Server Affinity for Client Connections

A system administrator can establish load balancing of JMS destinations across multiple servers in a cluster by configuring multiple JMS servers and using targets to assign them to the defined WebLogic Servers. Each JMS server is deployed on exactly one WebLogic Server and handles requests for a set of destinations. During the configuration phase, the system administrator enables load balancing by specifying targets for JMS servers. For instructions on setting up targets, see Configure Migratable Targets for Pinned Services. For instructions on deploying a JMS server to a migratable target, see Deploying_ Activating_ and Migrating Migratable Services.

A system administrator can establish cluster-wide, transparent access to destinations from any server in the cluster by configuring multiple connection factories and using targets to assign them to WebLogic Servers. Each connection factory can be deployed on multiple WebLogic Servers. For more information about connection factories, see ConnectionFactory in *Developing JMS Applications for Oracle WebLogic Server*.

The application uses the JNDI to look up a connection factory and create a connection to establish communication with a JMS server. Each JMS server handles requests for a set of destinations. Requests for destinations not handled by a JMS server are forwarded to the appropriate server.

WebLogic Server provides server affinity for client connections. If an application has a connection to a given server instance, JMS will attempt to establish new JMS connections to the same server instance.

When creating a connection, JMS will try first to achieve initial context affinity. It will attempt to connect to the same server or servers to which a client connected for its initial context, assuming that the server instance is configured for that connection factory. For example, if the connection factory is configured for servers A and B, but the client has an InitialContext on server C, then the connection factory will not establish the new connection with A, but will choose between servers B and C.

If a connection factory cannot achieve initial context affinity, it will try to provide affinity to a server to which the client is already connected. For instance, assume the client has an initial context on server A and some other type of connection to server B. If the client uses a connection factory configured for servers B and C, it will not achieve initial context affinity. The connection factory will instead attempt to achieve server affinity by trying to create a connection to server B, to which it already has a connection, rather than server C.

If a connection factory cannot provide either initial context affinity or server affinity, then the connection factory is free to make a connection wherever possible. For instance, assume a client has an initial context on server A, no other connections and a connection factory configured for servers B and C. The connection factory is unable to provide any affinity and is free to attempt new connections to either server B or C.

> **Note**
>
> In the last case, if the client attempts to make a second connection using the same connection factory, it will go to the same server as it did on the first attempt. That is, if it chose server B for the first connection, when the second connection is made, the client will have a connection to server B and the server affinity rule will be enforced.

5

Failover and Replication in a Cluster

Learn how Oracle WebLogic Server detects failures in a cluster and how failover is accomplished for different types of objects.

This chapter focuses on failover and replication at the application level. WebLogic Server also supports automatic migration of server instances and services after failure. For more information, see Whole Server Migration.

This chapter includes the following sections:

How WebLogic Server Detects Failures

WebLogic Server instances in a cluster detect failures of their peer server instances by monitoring socket connections to a peer server and regular server heartbeat messages.

Failure Detection Using IP Sockets

WebLogic Server instances monitor the use of IP sockets between peer server instances as an immediate method of detecting failures. If a server connects to one of its peers in a cluster and begins transmitting data over a socket, an unexpected closure of that socket causes the peer server to be marked as "failed," and its associated services are removed from the JNDI naming tree.

The WebLogic Server "Heartbeat"

If clustered server instances do not have opened sockets for peer-to-peer communication, failed servers may also be detected via the WebLogic Server heartbeat. All server instances in a cluster use multicast or unicast to broadcast regular server heartbeat messages to other members of the cluster.

Each heartbeat message contains data that uniquely identifies the server that sends the message. Servers broadcast their heartbeat messages at regular intervals of 10 seconds. In turn, each server in a cluster monitors the multicast or unicast address to ensure that all peer server's heartbeat messages are being sent.

If a server monitoring the multicast or unicast address misses three heartbeats from a peer server (for example, if it does not receive a heartbeat from the server for 30 seconds or longer), the monitoring server marks the peer server as "failed". If necessary, it updates its local JNDI tree to retract the services hosted on the failed server.

In this way, servers can detect failures even if they have no sockets open for peer-to-peer communication.

> **Note**
>
> The default cluster messaging mode is unicast.
>
> For more information about how WebLogic Server uses IP sockets and either multicast or unicast communications, see Communications In a Cluster.

Replication and Failover for Servlets and JSPs

To support automatic replication and failover for servlets and JSPs within a cluster, Weblogic Server supports two mechanisms for preserving HTTP session state: hardware load balancers and proxy plug-ins.

- Hardware load balancers

  For clusters that use a supported hardware load balancing solution, the load balancing hardware simply redirects client requests to any available server in the WebLogic Server cluster. The cluster itself obtains the replica of the client's HTTP session state from a secondary server in the cluster.

- Proxy plug-ins

  For clusters that use a supported Web server and WebLogic plug-in, the plug-in redirects client requests to any available server instance in the WebLogic Server cluster. The cluster obtains the replica of the client's HTTP session state from either the primary or secondary server instance in the cluster.

See the following sections:

HTTP Session State Replication

WebLogic Server provides three methods for replicating HTTP session state across clusters:

- In-memory replication

  Using in-memory replication, WebLogic Server copies a session state from one server instance to another. The primary server creates a primary session state on the server to which the client first connects, and a secondary replica on another WebLogic Server instance in the cluster. The replica must be up-to-date, so you can use it if the server that hosts the servlet fails.

- JDBC-based persistence

  In JDBC-based persistence, WebLogic Server maintains the HTTP session state of a servlet or JSP using file-based or JDBC-based persistence. For more information about these persistence mechanisms, see Configuring Session Persistence in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

  JDBC-based persistence is also used for HTTP session state replication within a Wide Area Network (WAN). See WAN HTTP Session State Replication.

> **Note**
>
> Web applications which have persistent store type set to `replicated` or `replicated_if_clustered` will have to be targeted to the cluster or all the nodes of that cluster. If it is targeted to only some nodes in the cluster, the Web application will not be deployed. In-memory replication *requires* that Web applications be deployed homogeneously on all the nodes in a cluster.

- Coherence*Web

  You can use Coherence*Web for session replication. Coherence*Web is not a replacement for WebLogic Server's in-memory HTTP state replication services. However, you should consider using Coherence*Web when an application has large HTTP session state objects, when running into memory constraints due to storing HTTP session object data, or if you want to reuse an existing Coherence cluster.

  For more information, see Using Coherence*Web with WebLogic Server in *Administering HTTP Session Management with Oracle Coherence*Web*.

The following sections describe session state replication using in-memory replication.

Requirements for HTTP Session State Replication

To use in-memory replication for HTTP session states, you must access the WebLogic Server cluster using either a collection of Web servers with identically configured WebLogic proxy plug-ins, or load balancing hardware.

Supported Server and Proxy Software

The WebLogic proxy plug-in maintains a list of WebLogic Server instances that host a clustered servlet or JSP, and forwards HTTP requests to those instances using a round-robin strategy. The plug-in also provides the logic necessary to locate the replica of a client's HTTP session state if a WebLogic Server instance should fail.

In-memory replication for HTTP session state is supported by the following Web servers and proxy software:

- WebLogic Server with the `HttpClusterServlet`.
- Apache with the Apache HTTP Server (proxy) plug-in.
- Oracle HTTP Server with the Oracle HTTP Server (proxy) plug-in.
- Microsoft Internet Information Server with the WebLogic Server Microsoft-IIS proxy plug-in.

For instructions on setting up proxy plug-ins, see Configure Proxy Plug-Ins.

Load Balancer Requirements

If you choose to use load balancing hardware instead of a proxy plug-in, it must support a compatible passive or active cookie persistence mechanism, and SSL persistence. For details on these requirements, see Load Balancer Configuration Requirements. For instructions on setting up a load balancer, see Configuring Load Balancers that Support Passive Cookie Persistence.

Programming Considerations for Clustered Servlets and JSPs

This section highlights key programming constraints and recommendations for servlets and JSPs that you will deploy in a clustered environment.

- Session Data Must Be Serializable

  To support in-memory replication of HTTP session states, all servlet and JSP session data must be serializable.

  > **Note**
  >
  > Serialization is the process of converting a complex data structure, such as a parallel arrangement of data (in which a number of bits are transmitted at a time along parallel channels) into a serial form (in which one bit at a time is transmitted); a serial interface provides this conversion to enable data transmission.

  Every field in an object must be serializable or transient in order to consider the object serializable. If the servlet or JSP uses a combination of serializable and non-serializable objects, WebLogic Server does not replicate the session state of the non-serializable objects.

- Use `setAttribute` to Change Session State

  In an HTTP servlet that implements `javax.servlet.http.HttpSession`, use `HttpSession.setAttribute` (which replaces the deprecated `putValue`) to change attributes in a session object. If you set attributes in a session object with `setAttribute`, the object and its attributes are replicated in a cluster using in-memory replication. If you use other set methods to change objects within a session, WebLogic Server does not replicate those changes. Every time a change is made to an object that is in the session, `setAttribute()` should be called to update that object across the cluster.

  Likewise, use `removeAttribute` (which, in turn, replaces the deprecated `removeValue`) to remove an attribute from a session object.

  > **Note**
  >
  > Use of the deprecated `putValue` and `removeValue` methods will also cause session attributes to be replicated.

- Consider Serialization Overhead

  Serializing session data introduces some overhead for replicating the session state. The overhead increases as the size of serialized objects grow. If you plan to create very large objects in the session, test the performance of your servlets to ensure that performance is acceptable.

- Control Frame Access to Session Data

  If you are designing a Web application that utilizes multiple frames, keep in mind that there is no synchronization of requests made by frames in a given frameset. For example, it is possible for multiple frames in a frameset to create multiple sessions on behalf of the client application, even though the client should logically create only a single session.

In a clustered environment, poor coordination of frame requests can cause unexpected application behavior. For example, multiple frame requests can "reset" the application's association with a clustered instance, because the proxy plug-in treats each request independently. It is also possible for an application to corrupt session data by modifying the same session attribute by multiple frames in a frameset.

To avoid unexpected application behavior, carefully plan how you access session data with frames. You can apply one of the following general rules to avoid common problems:

- In a given frameset, ensure that only one frame creates and modifies session data.
- Always create the session in a frame of the first frameset your application uses (for example, create the session in the first HTML page that is visited). After the session has been created, access the session data only in framesets other than the first frameset.

Using Replication Groups

By default, WebLogic Server attempts to create session state replicas on a different machine than the one that hosts the primary session state. You can further control where secondary states are placed using replication groups. A replication group is a preferred list of clustered servers to be used for storing session state replicas.

Using the WebLogic Remote Console or FMWC, you can define unique machine names that will host individual server instances. These machine names can be associated with new WebLogic Server instances to identify where the servers reside in your system.

Machine names are generally used to indicate servers that run on the same machine. For example, you would assign the same machine name to all server instances that run on the same machine, or the same server hardware.

If you do not run multiple WebLogic Server instances on a single machine, you do not need to specify WebLogic Server machine names. Servers without a machine name are treated as though they reside on separate machines. For detailed instructions on setting machine names, see Configure Machine Names.

When you configure a clustered server instance, you can assign the server to a replication group, and a preferred secondary replication group for hosting replicas of the primary HTTP session states created on the server.

When a client attaches to a server in the cluster and creates a primary session state, the server hosting the primary state ranks other servers in the cluster to determine which server should host the secondary. Server ranks are assigned using a combination of the server's location (whether or not it resides on the same machine as the primary server) and its participation in the primary server's preferred replication group.

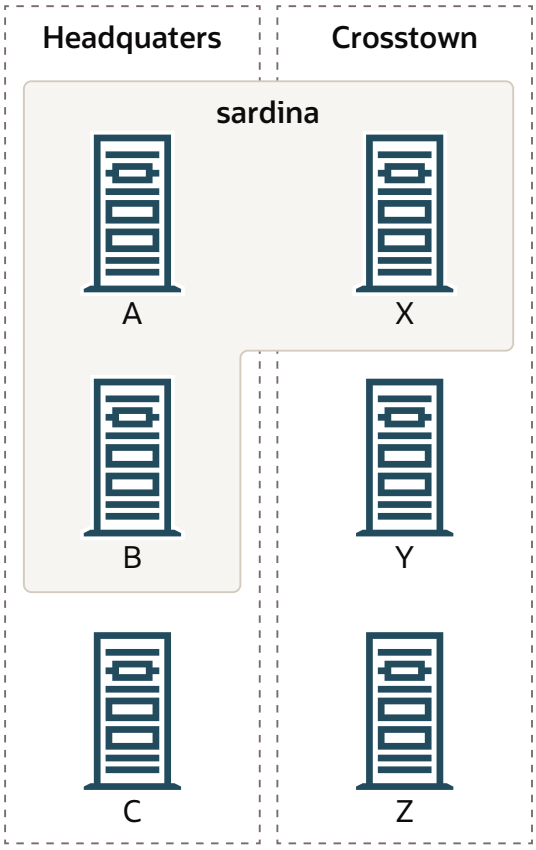
Table 5-1 shows the relative ranking of servers in a cluster.

Table 5-1 Ranking Server Instances for Session Replication

Server Rank	Server Resides on a Different Machine	Server is a Member of Preferred Replication Group
1	Yes	Yes
2	No	Yes
3	Yes	No
4	No	No

Using these rules, the primary WebLogic Server ranks other members of the cluster and chooses the highest-ranked server to host the secondary session state. For example, Figure 5-1 shows replication groups configured for different geographic locations.

Figure 5-1 Replication Groups for Different Geographic Locations



In this example, Servers A, B, and C are members of the replication group "Headquarters" and use the preferred secondary replication group "Crosstown." Conversely, Servers X, Y, and Z are members of the "Crosstown" group and use the preferred secondary replication group "Headquarters." Servers A, B, and X reside on the same machine, "sardina."

If a client connects to Server A and creates an HTTP session state, Servers will be as follows:

- Servers Y and Z are most likely to host the replica of this state since they reside on separate machines and are members of Server A's preferred secondary group.
- Server X holds the next-highest ranking because it is also a member of the preferred replication group (even though it resides on the same machine as the primary.)
- Server C holds the third-highest ranking since it resides on a separate machine but is not a member of the preferred secondary group.
- Server B holds the lowest ranking, because it resides on the same machine as Server A (and could potentially fail along with A if there is a hardware failure) and it is not a member of the preferred secondary group.

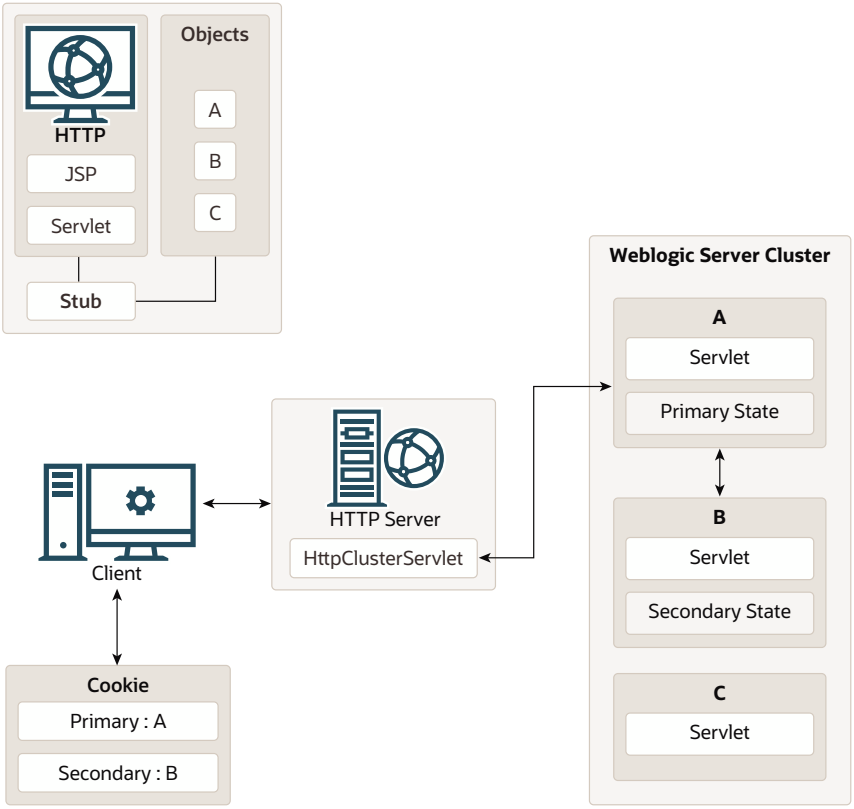
For instructions to configure a server's membership in a replication group, or to assign a server's preferred secondary replication group, see Configure Replication Groups.

Accessing Clustered Servlets and JSPs Using a Proxy

This section describes the connection and failover processes for requests that are proxied to clustered servlets and JSPs. For instructions on setting up proxy plug-ins, see Configure Proxy Plug-Ins.

Figure 5-2 depicts a client accessing a servlet hosted in a cluster. This example uses a single WebLogic Server instance to serve static HTTP requests only; all servlet requests are forwarded to the WebLogic Server cluster via the `HttpClusterServlet`.

Figure 5-2 Accessing Servlets and JSPs using a Proxy



> **Note**
>
> The discussion that follows also applies if you use a third-party Web server and WebLogic proxy plug-in, rather than WebLogic Server and the `HttpClusterServlet`.

Proxy Connection Procedure

When the HTTP client requests the servlet, `HttpClusterServlet` proxies the request to the WebLogic Server cluster. `HttpClusterServlet` maintains the list of all servers in the cluster, and the load balancing logic to use when accessing the cluster. In the above example, `HttpClusterServlet` routes the client request to the servlet hosted on WebLogic Server A. WebLogic Server A becomes the primary server hosting the client's servlet session.

To provide failover services for the servlet, the primary server replicates the client's servlet session state to a secondary WebLogic Server in the cluster. This ensures that a replica of the session state exists even if the primary server fails (for example, due to a network failure). In the example above, Server B is selected as the secondary.

The servlet page is returned to the client through the `HttpClusterServlet`, and the client browser is instructed to write a cookie that lists the primary and secondary locations of the servlet session state. If the client browser does not support cookies, WebLogic Server can use URL rewriting instead.

Using URL Rewriting to Track Session Replicas

In its default configuration, WebLogic Server uses client-side cookies to keep track of the primary and secondary server that host the client's servlet session state. If client browsers have disabled cookie usage, WebLogic Server can also keep track of primary and secondary servers using URL rewriting. With URL rewriting, both locations of the client session state are embedded into the URLs passed between the client and proxy server. To support this feature, you must ensure that URL rewriting is enabled on the WebLogic Server cluster. For instructions on how to enable URL rewriting, see Using URL Rewriting Instead of Cookies in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

Proxy Failover Procedure

When the primary server fails, `HttpClusterServlet` use the client's cookie information to determine the location of the secondary WebLogic Server that hosts the replica of the session state. `HttpClusterServlet` automatically redirects the client's next HTTP request to the secondary server and failover is transparent to the client.

After the failure, WebLogic Server B becomes the primary server hosting the servlet session state, and a new secondary is created (Server C in the previous example). In the HTTP response, the proxy updates the client's cookie to reflect the new primary and secondary servers, to account for the possibility of subsequent failovers.

> **Note**
>
> Now WebLogic proxy plug-ins randomly pick up a secondary server after the failover.

In a two-server cluster, the client would transparently failover to the server hosting the secondary session state. However, replication of the client's session state would not continue unless another WebLogic Server became available and joined the cluster. For example, if the original primary server was restarted or reconnected to the network, it would be used to host the secondary session state.

Accessing Clustered Servlets and JSPs with Load Balancing Hardware

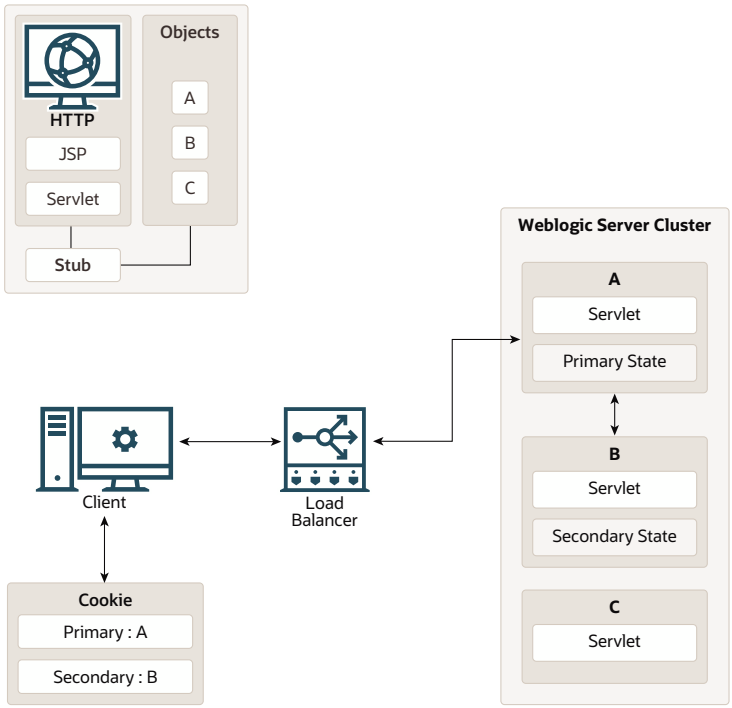
To support direct client access via load balancing hardware, the WebLogic Server replication system allows clients to use secondary session states regardless of the server to which the client fails over. WebLogic Server uses client-side cookies or URL rewriting to record primary and secondary server locations. However, this information is used only as a history of the servlet session state location; when accessing a cluster via load balancing hardware, clients do not use the cookie information to actively locate a server after a failure.

The following sections describe the connection and failover procedure when using HTTP session state replication with load balancing hardware.

Connection with Load Balancing Hardware

Figure 5-3 illustrates the connection procedure for a client accessing a cluster through a load balancer.

Figure 5-3 Connection with Load Balancing Hardware



When the client of a Web application requests a servlet using a public IP address:

1. The load balancer routes the client's connection request to a WebLogic Server cluster in accordance with its configured policies. It directs the request to WebLogic Server A.
2. WebLogic Server A acts as the primary host of the client's servlet session state. It uses the ranking system described in Using Replication Groups to select a server to host the replica of the session state. In the example above, WebLogic Server B is selected to host the replica.
3. The client is instructed to record the location of WebLogic Server instances A and B in a local cookie. If the client does not allow cookies, the record of the primary and secondary servers can be recorded in the URL returned to the client via URL rewriting.

> **Note**
>
> You must enable WebLogic Server URL rewriting capabilities to support clients that disallow cookies, as described in Using URL Rewriting to Track Session Replicas.

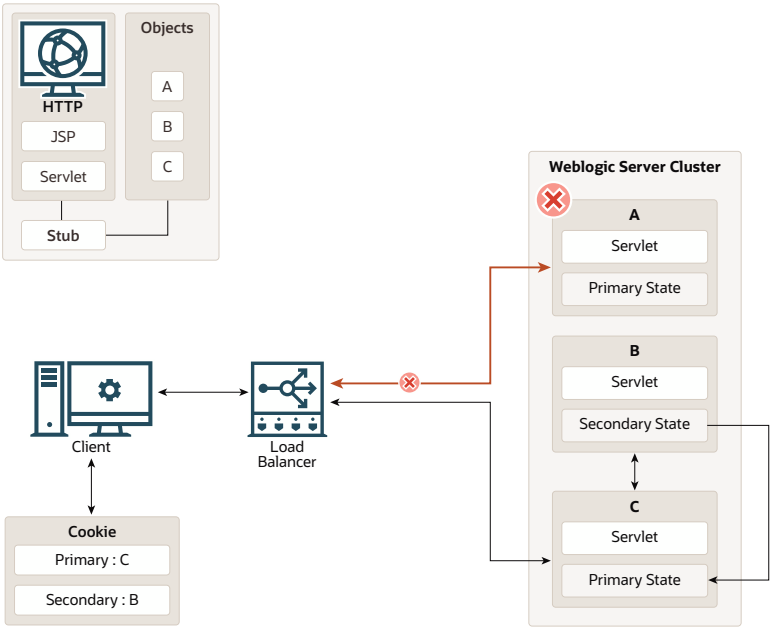
4. As the client makes additional requests to the cluster, the load balancer uses an identifier in the client-side cookie to ensure that those requests continue to go to WebLogic Server A (rather than being load-balanced to another server in the cluster). This ensures that the

client remains associated with the server hosting the primary session object till the end of the session.

Failover with Load Balancing Hardware

If Server A fail during the client's session, the client's next connection request to Server A also fails, as illustrated in Figure 5-4.

Figure 5-4 Failover with Load Balancing Hardware



In response to the connection failure:

1. The load balancing hardware uses its configured policies to direct the request to an available WebLogic Server in the cluster. In the above example, assume that the load balancer routes the client's request to WebLogic Server C after WebLogic Server A fails.
2. When the client connects to WebLogic Server C, the server uses the information in the client's cookie (or the information in the HTTP request if URL rewriting is used) to acquire the session state replica on WebLogic Server B. The failover process remains completely transparent to the client.

WebLogic Server C becomes the new host for the client's primary session state, and WebLogic Server B continues to host the session state replica. This new information about the primary and secondary host is again updated in the client's cookie, or through the URL rewriting.

Session State Replication Across Clusters in a MAN/WAN

In addition to providing HTTP session state replication across servers within a cluster, WebLogic Server provides the ability to replicate HTTP session state across multiple clusters. This improves high-availability and fault tolerance by allowing clusters to be spread across multiple geographic regions, power grids, and Internet service providers.

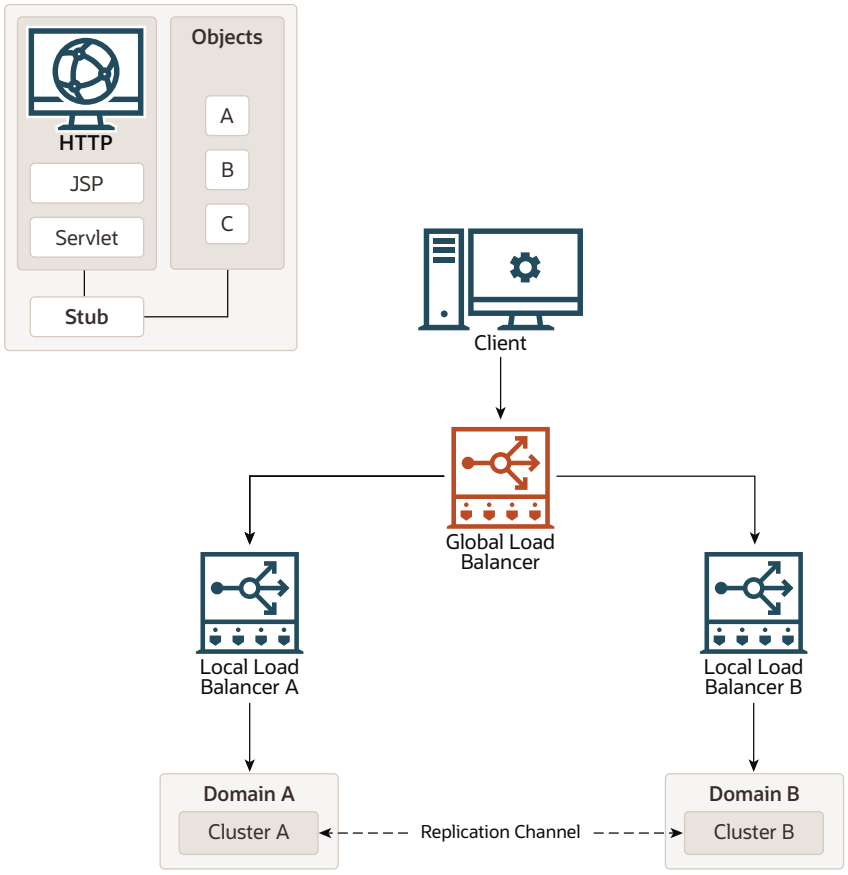
For general information on HTTP session state replication, see HTTP Session State Replication. For more information on using hardware load balancers, see Accessing Clustered Servlets and JSPs with Load Balancing Hardware.

The following sections discuss mechanisms for cross-cluster replication supported by WebLogic Server.

Network Requirements for Cross-cluster Replication

To perform cross-cluster replication with WebLogic Server, your network must include global and local hardware load balancers. Figure 5-5 shows how both types of load balancers interact within a multi-cluster environment to support cross-cluster replication. For general information on using load balancer within a WebLogic Server environment, see Connection with Load Balancing Hardware.

Figure 5-5 Load Balancer Requirements for Cross-cluster Replications



The following sections describe each of the components in this network configuration.

Global Load Balancer

In a network configuration that supports cross-cluster replication, the global load balancer is responsible for balancing HTTP requests across clusters. When a request comes in, the global load balancer determines which cluster need to be sent based on the current number of requests being handled by each cluster. Then the request is passed to the local load balancer for the chosen cluster.

Local Load Balancer

The local load balancer receives HTTP requests from the global load balancer. The local load balancer is responsible for balancing HTTP requests across servers within the cluster.

Replication

To replicate session data from one cluster to another, a replication channel must be configured to communicate session state information from the primary to the secondary cluster. The specific method used to replicate session information depends on which type of cross-cluster replication you are implementing. See MAN HTTP Session State Replication or WAN HTTP Session State Replication.

Failover

When a server within a cluster fails, the local load balancer is responsible for transferring the request to other servers within a cluster. When the entire cluster fails, the local load balancer returns HTTP requests back to the global load balancer. The global load balancer then redirects this request to the other local load balancer.

Configuration Requirements for Cross-Cluster Replication

The following procedures outline the basic steps required to configure cross-cluster replication.

1. Install WebLogic Server according to your network configuration and requirements. This includes installing a WebLogic Server instance on every physical machine that hosts a WebLogic Server instance.
2. Install and configure the hardware load balancers. See Network Requirements for Cross-cluster Replication. For more information on configuring load balancers, see Load Balancer Configuration.

   Following are some general considerations when configuring hardware load balancers to support cross-cluster replications:

   - You must configure your load balancer to maintain session IDs. If the load balancers do not maintain session ID, subsequent requests will always be sent to a new server. See Connection with Load Balancing Hardware.
   - You should ensure that the cluster failover timeout value is not set to high. This value should be around 3-5 seconds. Some hardware load balancers have default values that are much longer.
   - You must configure your load balancer to know which backup cluster to use when a primary cluster or server fails.
3. Create and configure your domains according to your cluster requirements.

   > **Note**
   >
   > Cross-cluster replication requires that each cluster be assigned to a different domain.

   In addition to creating and configuring your domains, you should also create and configure your clusters and Managed Servers. For information about creating and configuring domains, clusters, and Managed Servers, see the following topics:

- Understanding Oracle WebLogic Server Domains in *Understanding Domain Configuration for Oracle WebLogic Server*.
- Clusters in *Creating WebLogic Domains Using the Configuration Wizard*.
- Each domain should be set up and configured identically. In addition to identical domain, cluster and server configuration, the number of servers clusters, etc. should be identical.

Following are some considerations when configuring domains to support cross-cluster replication:

- Application deployment should be identical in each domain.
- When setting up your domains, you must enable trust between both domains. See Enabling Trust Between WebLogic Server Domains in *Administering Security for Oracle WebLogic Server*.

4. If you are using cross-cluster replication in a WAN environment, you must create a data source that is used to maintain session state. See Database Configuration for WAN Session State Replication.
5. After you create and configure your domains, servers, and clusters, you should verify whether the configuration elements specific to cross-cluster replication have been configured correctly. These parameters must be configured identically for both domains.

   Table 5-2 lists the subelements of the cluster element in `config.xml` that are used to configure cross-cluster replication.

Table 5-2 Cluster Elements in config.xml

Element	Description
cluster-type	This setting must match the replication type you are using and must be consistent across both clusters. The valid values are `man` or `wan` .
remote-cluster-address	This is the address used to communicate replication information to the other cluster. This should be configured so that communications between clusters do not go through a load balancer.
replication-channel	This is the network channel used to communicate replication information to the other cluster. **Note:** The named channel must exist on all members of the cluster and must be configured to use the same protocol. The selected channel may be configured to use a secure protocol.
data-source-for-session-persistence	This is the data source that is used to store session information when using JDBC-based session persistence. This method of session state replication is used to perform cross-cluster replication within a WAN. See Database Configuration for WAN Session State Replication.
session-flush-interval	This is the interval, in seconds, the cluster waits to flush HTTP sessions to the backup cluster.
session-flush-threshold	If the number of HTTP sessions reaches the value of session-flush-threshold, the sessions are flushed to the backup cluster. This allows servers to flush sessions faster under heavy loads.

Table 5-2 (Cont.) Cluster Elements in config.xml

Element	Description
inter-cluster-comm-link-health-check-interval	This is the amount of time, in milliseconds, between consecutive checks to determine if the link between two clusters is restored.

Configuring Session State Replication Across Clusters

You can use a third-party replication product to replicate state across clusters or allow WebLogic Server to replicate session state across clusters. The following configuration considerations should be kept in mind depending on which method you use:

- If you are using a third-party product, ensure that you have specified a value for `data-source-for-session-persistence`, and that `remote-cluster-address` is blank.
- If you are using WebLogic Server to handle session state replication, you must configure both the `data-source-for-session-persistence` and the `remote-cluster-address`.

If `remote-cluster-address` is NULL, WebLogic Server assumes that you are using a third-party product to handle replication. In this case, session data does not persist in the remote database but the local.

Configuring a Replication Channel

A replication channel is a normal network channel that is dedicated specifically to replicating traffic between clusters. See Configuring Network Resources in *Administering Server Environments for Oracle WebLogic Server*.

When creating a network channel to be used as the replication channel in cross-cluster replication, the following considerations apply:

- You must ensure that the replication channel is created on all cluster members and has the same name.
- The channel should be used only for replication. Other types of network traffic should be directed to other network channels.

MAN HTTP Session State Replication

Resources within a metropolitan area network (MAN) are often in physically separate locations, but are geographically close enough that network latency is not an issue. Network communication in a MAN generally has low latency and fast interconnect. Clusters within a MAN can be installed in physically separate locations which improves availability.

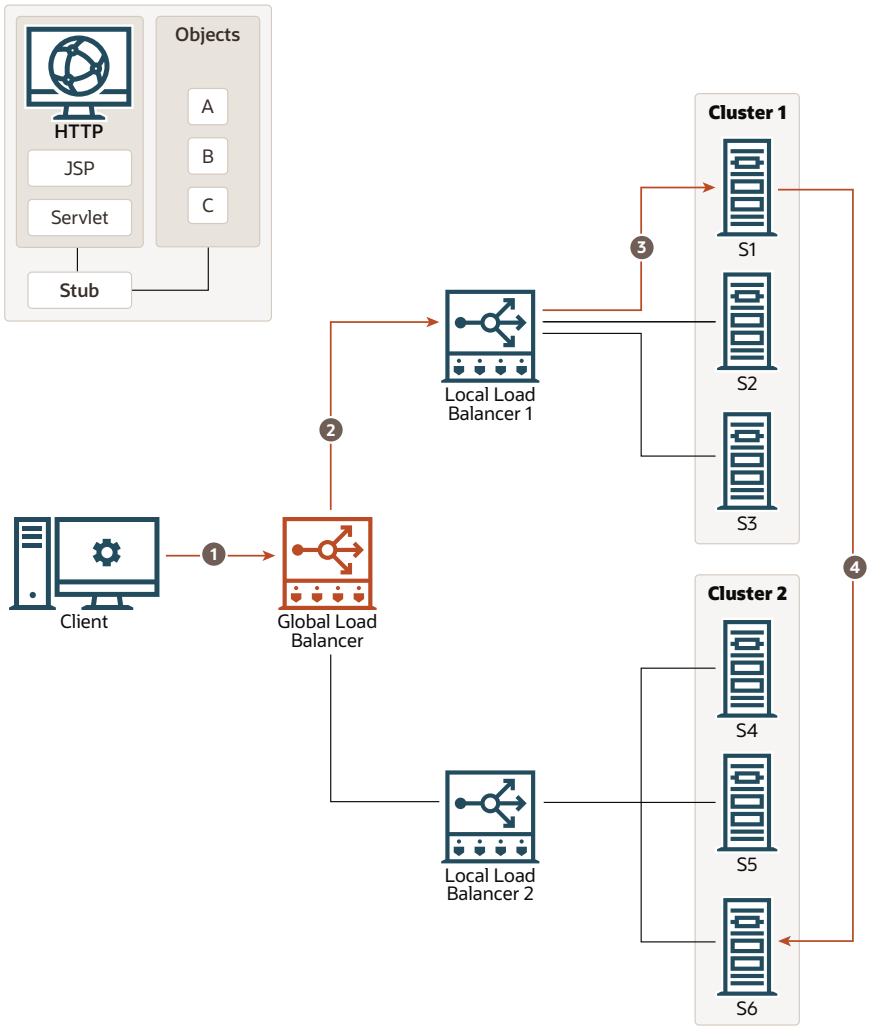
To provide failover within a MAN, WebLogic Server provides an in-memory mechanism that works between two separate clusters. This allows session state to be replicated synchronously from one cluster to another, provided that the network latency is a few milliseconds. The advantage of using a synchronous method is that reliability of in-memory replication is guaranteed.

> **Note**
>
> The performance of synchronous state replication is dependant on the network latency between clusters. You should use this method only if the network latency between the clusters is tolerable.

Replication Within a MAN

This section discusses possible failover scenarios across multiple clusters within a MAN. Figure 5-6 shows a typical multi-cluster environment within a MAN.

Figure 5-6 MAN Replication



This figure shows the following HTTP session state scenario:

1. A client makes a request which passes through the global load balancer.
2. The global load balancer passes the request to a local load balancer based on current system load. In this case, the session request is passed to Local Load Balancer 1.

3. The local load balancer in turn passes the request to a server within a cluster based on system load, in this case S1. Once the request reaches S1, this Managed Server becomes the primary server for this HTTP session. This server will handle subsequent requests assuming there are no failures.
4. Session state information is stored in the database of the primary cluster.
5. After the server establishes the HTTP session, the current session state is replicated to the designated secondary server.

Failover Scenarios in a MAN

The following sections describe various failover scenarios based on the MAN configuration in Figure 5-6.

Failover Scenario 1

If all of the servers in Cluster 1 fail, the global load balancer will automatically fail all subsequent session requests to Cluster 2. All sessions that have been replicated to Cluster 2 will be recovered and the client will experience no data loss.

Failover Scenario 2

Assume that the primary server S1 is being hosted on Cluster 1, and the secondary server S6 is being hosted on Cluster 2. If S1 crashes, then any other server in Cluster 1 (S2 or S3) can pick up the request and retrieve the session data from server S6. S6 will continue to be the backup server.

Failover Scenario 3

Assume that the primary server S1 is being hosted on Cluster 1, and the secondary server S6 is being hosted on Cluster 2. If the secondary server S6 fails, then the primary server S1 will automatically select a new secondary server on Cluster 2. Upon receiving a client request, the session information will be backed up on the new secondary server.

Failover Scenario 4

If the communication between the two clusters fails, the primary server will automatically replicate session state to a new secondary server within the local cluster. After the communication between the two clusters, any subsequent client requests will replicate on the remote cluster.

MAN Replication, Load Balancers, and Session Stickiness

MAN replication relies on global load balancers to maintain cluster affinity and local load balancers to maintain server affinity. If a server within a cluster fails, the local load balancer is responsible for ensuring that session state is replicated to another server in the cluster. If all the servers within a cluster fail or are unavailable, the global load balancer is responsible for replicating session state to another cluster. This ensures that failover to another cluster does not occur unless the entire cluster fails.

Once a client establishes a connection through a load balancer to a cluster, the client must maintain stickiness to that cluster as long as it is healthy.

WAN HTTP Session State Replication

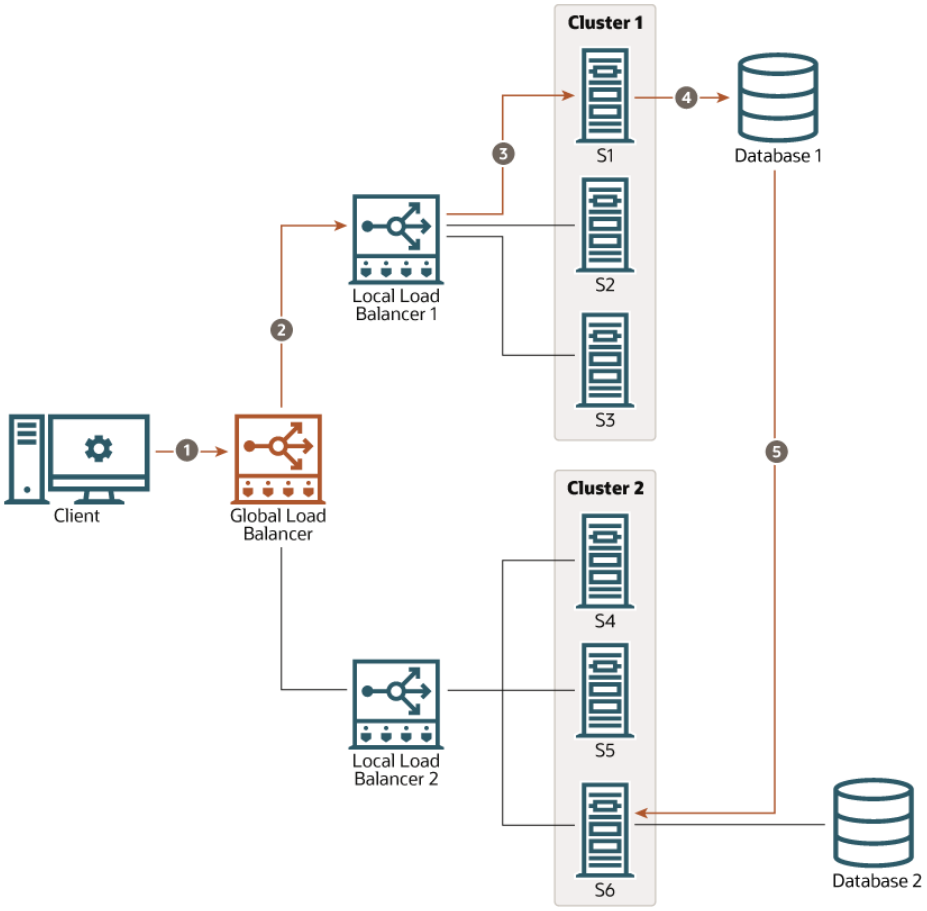
Resources in a wide area network (WAN) are frequently spread across separate geographical regions. In addition to the requirements of network traffic to cross long distances, these resources are often separated by multiple routers and other network bottlenecks. Network communication in a WAN generally has higher latency and slower interconnect.

Slower network performance within a WAN makes it difficult to use a synchronous replication mechanism like the one used within a MAN. WebLogic Server provides failover across clusters in WAN by using an asynchronous data replication scheme.

Replication Within a WAN

This section discusses possible failover scenarios across multiple clusters within a WAN. Figure 5-7 shows a typical multi-cluster environment within a WAN.

Figure 5-7 WAN Replication



This figure demonstrates the following HTTP session state scenario:

1. A client makes a request which passes through the global load balancer.
2. The global load balancer passes the request to a local load balancer based on current system load. In this case, the session request is passed to Local Load Balancer 1.
3. The local load balancer in turn passes the request to a server within a cluster based on system load, in this case S1. Once the request reaches S1, this Managed Server becomes the primary server for this HTTP session. This server will handle subsequent requests assuming there are no failures.
4. Session state information is stored in the database of the primary cluster.
5. After the server establishes the HTTP session, the current session state is replicated to the designated secondary server.

Failover Scenarios Within a WAN

This section describes the failover scenario within a WAN environment.

Failover Scenario

If all the servers in Cluster 1 fail, the global load balancer will automatically fail all subsequent session requests to Cluster 2. All sessions will be backed up according to the last known flush to the database.

Database Configuration for WAN Session State Replication

This section describes the data source configuration requirements for cross-cluster session state replication in a WAN. For more general information about setting up cross-cluster replication, see Configuration Requirements for Cross-Cluster Replication.

To enable cross-cluster replication within a WAN environment, you must create a JDBC data source that points to the database where session state information is stored. Perform the following procedures to setup and configure your database:

1. Install and configure your database server software according to your vendor's documentation.
2. Create a JDBC data source that references this database. For more information, see Configuring JDBC Data Sources in *Administering JDBC Data Sources for Oracle WebLogic Server*.

   This data source can also be configured as a JDBC Multi Data Source. For more information, see Configuring JDBC Multi Data Sources in *Administering JDBC Data Sources for Oracle WebLogic Server*.
3. Set the `DataSourceForSessionPersistence` for both the primary and secondary cluster to point to this data source.
4. Create a table called `WLS_WAN_PERSISTENCE` in your database according to the following schema:

```
CREATE TABLE WLS_WAN_PERSISTENCE_TABLE (
  WL_ID VARCHAR2(100) NOT NULL,
  WL_CONTEXT_PATH VARCHAR2(50) NOT NULL,
  WL_CREATE_TIME NUMBER(20),
  WL_ACCESS_TIME NUMBER(20),
  WL_MAX_INACTIVE_INTERVAL NUMBER(38),
  WL_VERSION NUMBER(20) NOT NULL,
  WL_INTERNAL_ATTRIBUTE NUMBER(38),
  WL_SESSION_ATTRIBUTE_KEY VARCHAR2(100),
  WL_SESSION_ATTRIBUTE_VALUE LONG RAW,
  PRIMARY KEY(WL_ID, WL_CONTEXT_PATH,
  WL_VERSION, WL_SESSION_ATTRIBUTE_KEY));
```

   Table 5-3 describes each row in the Replication table.

Table 5-3 Contents of Replication Table

Database Row	Description
`wl_id`	Stores the HTTP session ID.
`wl_context_path`	Stores the context path to the Web application that created the session.
`wl_create_time`	Stores the time the session state was created.

Table 5-3 (Cont.) Contents of Replication Table

Database Row	Description
`wl_session_values`	Stores the session attributes.
`wl_access_time`	Stores the time of the last update to the session state.
`wl_max_inactive_interval`	Stores the `MaxInactiveInterval` of the session state.
`wl_version`	Stores the version of the session. Each update to a session has an associated version.

Replication and Failover for EJBs and RMIs

For clustered EJBs and RMIs, failover is accomplished using the object's replica-aware stub. When a client makes a call through a replica-aware stub to a service that fails, the stub detects the failure and retries the call on another replica.

The key technology that supports object clustering in WebLogic Server is the *replica-aware stub*. With clustered objects, automatic failover generally occurs only in cases where the object is *idempotent*. An object is idempotent if any method can be called multiple times with no different effect than calling the method once. This is always true for methods that have no permanent side effects. Methods that do have side effects have to be written with idempotence in mind.

Consider a shopping cart service call `addItem()` that adds an item to a shopping cart. Suppose client C invokes this call on a replica on Server S1. After S1 receives the call, but before it successfully returns to C, S1 crashes. At this point the item has been added to the shopping cart, but the replica-aware stub has received an exception. If the stub has to retry the method on Server S2, the item would be added second time to the shopping cart. Because of this, replica-aware stubs will not attempt to retry a method that fails after the request is sent but before it returns. This behavior can be overridden by marking a service idempotent.

Clustering Objects with Replica-Aware Stubs

If an EJB or RMI object is clustered, instances of the object are deployed on all WebLogic Server instances in the cluster. The client has a choice about which instance of the object to call. Each instance of the object is referred to as a replica.

When you compile an EJB that supports clustering (as defined in its deployment descriptor), `appc` passes the EJB's interfaces through the `rmic` compiler to generate replica-aware stubs for the bean. For RMI objects, you generate replica-aware stubs explicitly using command-line options to `rmic`, as described in Using the WebLogic RMI Compiler in *Developing RMI Applications for Oracle WebLogic Server*.

A replica-aware stub appears to the caller as a normal RMI stub. Instead of representing a single object, however, the stub represents a collection of replicas. The replica-aware stub contains the logic required to locate an EJB or RMI class on any WebLogic Server instance on which the object is deployed. When you deploy a cluster-aware EJB or RMI object, its implementation is bound into the JNDI tree. As described in Cluster-Wide JNDI Naming Service, clustered WebLogic Server instances have the capability to update the JNDI tree to list all server instances on which the object is available. When a client accesses a clustered object, the implementation is replaced by a replica-aware stub, which is sent to the client.

The stub contains the load balancing algorithm (or the call routing class) used to load balance method calls to the object. On each call, the stub can employ its load algorithm to choose which replica to call. This provides load balancing across the cluster in a way that is transparent to the caller. To understand the load balancing algorithms available for RMI objects and EJBs, see Load Balancing for EJBs and RMI Objects. If a failure occurs during the call, the stub intercepts the exception and retries the call on another replica. This provides a failover that is also transparent to the caller.

Clustering Support for Different Types of EJBs

EJBs differ from plain RMI objects. In that, each EJB can potentially generate two different replica-aware stubs: one for the `EJBHome` interface and one for the `EJBObject` interface. This means that EJBs can potentially realize the benefits of load balancing and failover on two levels:

- When a client looks up an EJB object using the `EJBHome` stub.
- When a client makes method calls against the EJB using the `EJBObject` stub.

The following sections describe clustering support for different types of EJBs.

Clustered EJBHomes

All bean homes interfaces that are used to find or create bean instances can be clustered, by specifying the home-is-clusterable element in `weblogic-ejb-jar.xml`.

> **Note**
>
> Stateless session beans, stateful session beans, and entity beans have home interfaces, whereas, message-driven beans do not.

When a bean is deployed to a cluster, each server binds the bean's home interface to its cluster JNDI tree under the same name. When a client requests the bean's home from the cluster, the server instance that does the look-up returns a `EJBHome` stub that has a reference to the home on each server.

When the client issues a `create()` or `find()` call, the stub selects a server from the replica list in accordance with the load balancing algorithm, and routes the call to the home interface on that server. The selected home interface receives the call and creates a bean instance on that server instance and executes the call to create an instance of the bean.

> **Note**
>
> WebLogic Server supports load balancing algorithms that provide server affinity for EJB home interfaces. For information about understanding the server affinity and how it affects the load balancing and failover, see Round-Robin Affinity, Weight-Based Affinity, and Random-Affinity.

Clustered EJBObjects

An `EJBObject` stub tracks available replicas of an EJB in a cluster.

Stateless Session Beans

When a home creates a stateless bean, it returns an `EJBObject` stub that lists all of the servers in the cluster, to which the bean should be deployed. Because a stateless bean holds no state on behalf of the client, the stub is free to route any call to any server that hosts the bean. If the failure occurs, the stub will automatically failover. The stub does not automatically treat the bean as idempotent, so it will not recover automatically from all failures. If the bean has been written with idempotent methods, this can be noted in the deployment descriptor and automatic failover will be enabled in all cases.

> **Note**
>
> WebLogic Server supports load balancing options that provide server affinity for stateless EJB remote interfaces. For more information about understanding server affinity and how it affects load balancing and failover, see Round-Robin Affinity, Weight-Based Affinity, and Random-Affinity.

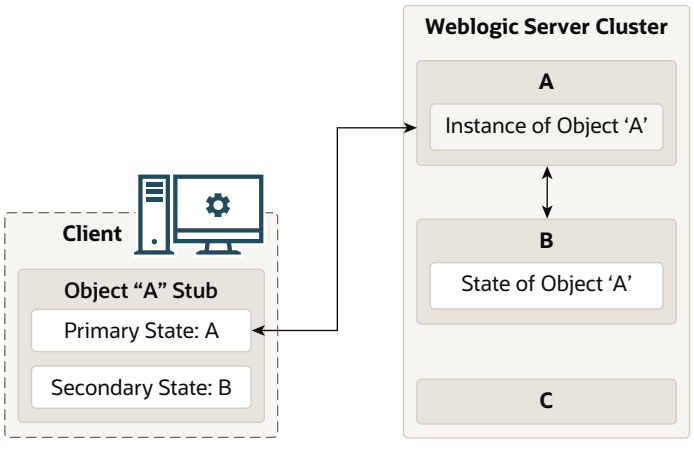
Stateful Session Beans

Method-level failover for a stateful service requires state replication. WebLogic Server satisfies this requirement by replicating the state of the primary bean instance to a secondary server instance, using a replication scheme similar to that used for HTTP session state.

When a home interface creates a stateless session bean instance, it selects a secondary instance to host the replicated state, using the same rules defined in Using Replication Groups. The home interface returns an `EJBObject` stub to the client that lists the location of the primary bean instance, and the location for the replicated bean state.

Figure 5-8 shows a client accessing a clustered stateful session EJB.

Figure 5-8 Client Accessing Stateful Session EJB



As the client makes changes to the state of the EJB, state differences are replicated to the secondary server instance. For EJBs that are involved in a transaction, replication occurs immediately after the transaction commits. For EJBs that are not involved in a transaction, replication occurs after each method invocation.

In both cases, only the actual changes to the EJB's state are replicated to the secondary server. This ensures that there is minimal overhead associated with the replication process.

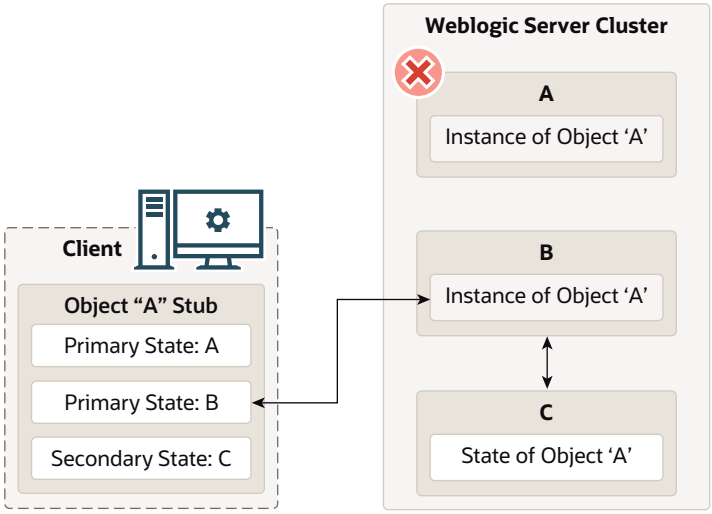
> **Note**
>
> The actual state of a stateful EJB is non-transactional, as described in the EJB specification. Although it is unlikely, there is a possibility that the current state of the EJB can be lost. For example, if a client commits a transaction involving the EJB and there is a failure of the primary server before the state change is replicated, the client will failover to the previously-stored state of the EJB.

Failover for Stateful Session EJBs

When the primary server fails, the client's EJB stub automatically redirects further requests to the secondary WebLogic Server instance. At this point, the secondary server creates a new EJB instance using the replicated state data, and the process continues on the secondary server.

After a failover, WebLogic Server chooses a new secondary server to replicate EJB session states (if another server is available in the cluster). The location of the new primary and secondary server instances are automatically updated in the client's replica-aware stub on the next method invocation, as shown in Figure 5-9.

Figure 5-9 Replica Aware Stubs are Updated after Failover



Clustering Support for RMI Objects

WebLogic RMI provides special extensions for building clustered remote objects. These are the extensions used to build the replica-aware stubs described in the EJB section. For more information about using RMI in clusters, see WebLogic RMI Features in *Developing RMI Applications for Oracle WebLogic Server*.

Object Deployment Requirements

If you are programming EJBs to use it in a WebLogic Server cluster, read the instructions in this section to understand the capabilities of different EJB types in a cluster. Then ensure that you enable clustering in the EJB's deployment descriptor. For more information about the XML deployment elements relevant for clustering, see weblogic-ejb-jar.xml Deployment Descriptor Reference in *Developing Jakarta Enterprise Beans Using Deployment Descriptors*, for Oracle WebLogic Server.

If you are developing either EJBs or custom RMI objects, refer to Using WebLogic JNDI in a Clustered Environment in *Developing JNDI Applications for Oracle WebLogic Server* for more information about understanding the implications of binding clustered objects in the JNDI tree.

Other Failover Exceptions

Even if a clustered object is not idempotent, WebLogic Server performs automatic failover in the case of a `ConnectException` or `MarshalException`. Either of these exceptions indicates that the object could not have been modified, and therefore there is no danger of causing data inconsistency by failing over to another instance.

6

Whole Server Migration

Learn about the different migration mechanisms supported by Oracle WebLogic Server. The following sections focus on whole server-level migration, where a migratable server instance and all of its services are migrated to a different physical machine upon failure. WebLogic Server also supports service-level migration, as well as replication and failover at the application level. For more information, see Service Migration and Failover and Replication in a Cluster.

This chapter includes the following sections:

Understanding Server and Service Migration

In a WebLogic Server cluster, most services are deployed homogeneously on all server instances in the cluster, enabling transparent failover from one server instance to another. In contrast, pinned services such as JMS and the JTA transaction recovery system are targeted at individual server instances within a cluster for these services, WebLogic Server supports failure recovery with *migration*, as opposed to failover.

Migration in WebLogic Server is the process of moving a clustered WebLogic Server instance or a component running on a clustered server instance elsewhere in the event of failure. In the case of whole server migration, the server instance is migrated to a different physical machine upon failure. In the case of service-level migration, the services are moved to a different server instance within the cluster. For more information, see Service Migration.

To make JMS and the JTA transaction system highly available, WebLogic Server provides migratable servers. Migratable servers provide automatic and manual migration at the server-level rather than the service-level.

When a migratable server becomes unavailable for any reason (for example, if it hangs, loses network connectivity, or its host machine fails), migration is automatic. Upon failure, a migratable server is automatically restarted on the same machine, if possible. If the migratable server cannot be restarted on the machine where it failed, it is migrated to another machine. In addition, an administrator can manually initiate migration of a server instance.

Migration Terminology

Learn server and service migration terms.

- Migratable server: A clustered server instance that migrates in its entirety, along with all the services it hosts. Migratable servers are intended to host pinned services, such as JMS servers and JTA transaction recovery servers, but migratable servers can also host clusterable services. All services that run on a migratable server are highly available.
- Whole server migration: A WebLogic Server instance to be migrated to a different physical machine upon failure, either manually or automatically.
- Service migration:

  - Manual Service Migration: The manual migration of pinned JTA and JMS-related services (for example, JMS server, SAF agent, path service, and custom store) after the host server instance fails. See Service Migration.
  - Automatic Service Migration: JMS-related services, singleton services, and the JTA Transaction Recovery Service can be configured to automatically migrate to another member server instance when a member server instance fails or is restarted. See Service Migration.
- Cluster leader: One server instance in a cluster, elected by a majority of the server instances, that is responsible for maintaining the leasing information. See Non-database Consensus Leasing.
- Cluster master: One server instance in a cluster that contains migratable servers acts as the *cluster master* and orchestrates the process of automatic server migration in the event of failure. Any Managed Server in a cluster can serve as the cluster master, whether it hosts pinned services or not. See Cluster Master Role in Whole Server Migration.
- Singleton master: A lightweight singleton service that monitors other services that can be migrated automatically. The server instance that currently hosts the singleton master is responsible for starting and stopping the migration tasks associated with each migratable service. See Singleton Master.
- Candidate machines: A user-defined list of machines within a cluster that can be a potential target for migration.
- Target machines: A set of machines that are designated as allowable or preferred hosts for migratable servers.
- Node Manager: A WebLogic Server utility used by the Administration Server or a standalone Node Manager client, to start and stop migratable servers. Node Manager is invoked by the cluster master to shut down and restart migratable servers, as necessary. For background information about Node Manager and how it fits into a WebLogic Server environment, see Node Manager Overview in *Administering Node Manager for Oracle WebLogic Server*.
- Lease table: A database table in which migratable servers persist their state, and which the cluster master monitors to verify the health and liveness of migratable servers. See Leasing.
- Administration Server: It is used to configure migratable servers and target machines, to obtain the run-time state of migratable servers, and to orchestrate the manual migration process.
- Floating IP address: An IP address that follows a server instance from one physical machine to another after migration.

Leasing

Leasing is the process WebLogic Server uses to manage services that are required to run on only one member of a cluster at a time.

Leasing ensures exclusive ownership of a cluster-wide entity. Within a cluster, there is a single owner of a lease. Additionally, leases can failover in case of server or cluster failure. This helps to avoid having a single point of failure.

Features That Use Leasing

The following WebLogic Server features use leasing:

- Automatic Whole Server Migration: It uses leasing to elect a cluster master. The cluster master is responsible for monitoring other cluster members and for restarting failed members hosted on other physical machines.

  Leasing ensures that the cluster master is always running, but is only running on one server instance at a time within a cluster. See Cluster Master Role in Whole Server Migration.
- Automatic Service Migration: JMS-related services, singleton services, and the JTA Transaction Recovery Service can be configured to automatically migrate from an unhealthy hosting server instance to a healthy active server instance with the help of the health monitoring subsystem. When the migratable target is migrated, the pinned service hosted by that target is also migrated. Migratable targets use leasing to accomplish automatic service migration. See Service Migration.
- Singleton Services: A singleton service is a service running within a cluster that is available on only one member of the cluster at a time. Singleton services use leasing to accomplish this. See Singleton Master.
- Job Scheduler: The Job Scheduler is a persistent timer that is used within a cluster. The Job Scheduler uses the timer master to load balance the timer across a cluster.

  This feature requires an external database to maintain failover and replication information. However, you can use the non-database version consensus leasing with the Job Scheduler. For more information, see Non-Database Consensus Leasing.

> **Note**
>
> Beyond basic configuration, most leasing functionality is handled internally by WebLogic Server.

Types of Leasing

WebLogic Server provides two types of leasing functionality, depending on your requirements and environment.

- High availability database leasing: This type of leasing requires a high availability database to store leasing information. For information on general requirements and configuration, see High Availability Database Leasing.
- Non-database consensus leasing: This type of leasing stores the leasing information in-memory within a cluster member. Non-database consensus leasing requires that all server instances in the cluster are started by Node Manager. See Non-Database Consensus Leasing.

Within a WebLogic Server installation, you can use only one type of leasing. Although it is possible to implement multiple features that use leasing within your environment, each must use the same kind of leasing.

When switching from one leasing type to another, you must restart the entire cluster, not just the Administration Server. Changing the leasing type cannot be done dynamically.

Determining Which Type of Leasing To Use

The following considerations will help you determine which type of leasing is appropriate for your WebLogic Server environment:

- High availability database leasing

Database leasing basis is useful in environments that are already invested in a high availability database, like Oracle Real Application Clusters (RAC), for features like JMS store recovery. The high availability database instance can also be configured to support leasing with minimal additional configuration. This is particularly useful if Node Manager is not running in the system.

- Non-database consensus leasing

  This type of leasing provides a leasing basis option (consensus) that does not require the use of a high availability database. This has a direct benefit in automatic whole server migration. Without the high availability database requirement, consensus leasing requires less configuration to enable automatic server migration.

  Consensus leasing requires Node Manager to be configured and running. Automatic whole server migration also requires Node Manager for IP migration and server restart on another machine. Hence, consensus leasing works well since it does not impose additional requirements, but instead takes away an expensive one.

> **Note**
>
> As a best practice, Oracle recommends configuring database leasing instead of consensus leasing.

High Availability Database Leasing

In this type of leasing, lease information is maintained within a table in a high availability database. To ensure that the leasing information is always available to the server instances, a high availability database is required. Each member of the cluster must be able to connect to the database to access leasing information, and to update and renew their leases. Server instances will fail if the database becomes unavailable and they are not able to renew their leases.

High availability database leasing is useful for customers who already have a high availability database within their clustered environment. Database leasing allows you to use leasing functionality without requiring Node Manager to manage server instances within your environment.

The following procedure outlines the steps required to configure your database for leasing.

1. Configure the database for server migration by creating the leasing table in the database. The database stores leasing information that is used to determine whether or not a server instance is running or needs to be migrated. The leasing table stores the machine-server associations that are used to enable server migration and/or automatic service migration.

   There are two methods for creating the leasing table:

   - Manually. To create the leasing table, you can use the SQL and table definition in the schema file located at `WL_HOME/server/db/db_type/leasing.ddl`, where *db_type* is the type of database.
   - Automatically, where WebLogic Server automatically detects that a leasing table is missing, detects the database type, and then finds and runs the appropriate default DDL file to create the table. To enable automatic leasing table creation, or to change the default table name, use the `ClusterMBean` configuration options described in Table 6-1.

> **Note**
>
> The leasing table should be stored in a reliable, highly available database. The server instances are only as reliable as the database. For experimental purposes, a regular database will suffice. For a production environment, only high availability databases such as Oracle RAC are recommended. If the database goes down, all the migratable servers or automatically migratable services will shut themselves down. Migratable servers or automatically migratable services are only as reliable as the database that is used to store the leasing table.

2. Setup and configure a data source. This data source should point to the database configured in the previous step.

> **Note**
>
> - XA data sources are not supported for database leasing.
> - Database leasing is supported only with a JDBC driver that implements `java.sql.Driver`. For example, `org.apache.derby.jdbc.ClientDriver`.
>
> It is not supported with JDBC drivers that implement `java.sql.DataSource`. For example, `org.apache.derby.jdbc.ClientDataSource` and `oracle.jdbc.replay.OracleDataSourceImpl`.

See Configuring JDBC Data Sources in *Administering JDBC Data Sources for Oracle WebLogic Server*.

Table 6-1 ClusterMBean Configuration Options for Database Leasing

ClusterMBean Attribute	Default	Description
MigrationBasis	database	Specifies the type of cluster leasing to use for server and service migration, either consensus or database. When this attribute is set to consensus, the default, a Node Manager instance must also be configured and running. To enable database leasing, set this attribute to database. Database leasing requires that you have a highly available database and that you set the DataSourceForAutomaticMigration attribute. Optionally, you can also set the following attributes: • AutoMigrationTableName • AutoMigrationTableCreationPolicy • AutoMigrationTableCreationDDLFile **Note** If you enable database leasing and you do not set AutoMigrationTableCreationPolicy to Always, then you need to manually create a database leasing table as described in Step 1.
DataSourceForAutomaticMigration	none	Specifies the data source that database leasing uses to contact the database. The attribute must point to an existing non-XA data source. This setting applies only if the MigrationBasis attribute is set to database.
AutoMigrationTableName	ACTIVE	Specifies the name of the table to use for database leasing. This setting applies only if the MigrationBasis attribute is set to database. **Note:** • If the AutoMigrationTableCreationPolicy attribute is set to Always, then the table name format must be specified in the form [[[mycatalog.]myschema.]mytablename; for example myschema.mytablename. Each period symbol in the format is significant, where myschema generally corresponds to the user name in many databases. • When no table name is specified, the table name defaults to ACTIVE, and the database implicitly determines the schema according to the JDBC data source user.

Table 6-1 (Cont.) ClusterMBean Configuration Options for Database Leasing

ClusterMBean Attribute	Default	Description
AutoMigrationTableCreationPolicy	Disabled	Controls the behavior of automatic leasing table creation for database leasing. This setting applies only if the MigrationBasis attribute is set to database. Valid values are: • Disabled: Disables automatic leasing table creation. The behavior is the same as manual table creation. Any singletons dependent on cluster leasing will fail to start unless the table defined by the AutoMigrationTableName already exists in the database. For information on manually creating this table, see Step 1. • Always: Force automatic leasing table creation if the leasing table is not found. The default table name is ACTIVE and can be customized using the AutoMigrationTableName attribute. The SQL used to create the table when it is not found can be customized using the AutoMigrationTableCreationDDLFile attribute. **Note**: The syntax for AutoMigrationTableName changes based on the value of this setting. See AutoMigrationTableName for details.
AutoMigrationTableCreationDDLFile	See Description	Specifies the absolute path of a custom DDL file to be used to create the automatic migration database table. This setting applies only when the following conditions are met: • The MigrationBasis attribute is set to database • The AutoMigrationTableCreationPolicy attribute is set to Always • The table name specified by the AutoMigrationTableName attribute does not already exist in the database Note the following: • If the above conditions apply and this attribute is not set, then the system tries to find and run a default DDL file for creating the leasing database table in your WebLogic Server installation directory at the location *WL_HOME*/server/db/*DB_TYPE*/leasing.ddl. • If the above conditions apply and this attribute is set to the location of a DDL file, then the system runs the SQL in the specified custom DDL file for creating the leasing database table. For examples of DDL files, look in your WebLogic Server installation directory under *WL_HOME*/server/db. • When automatically creating a leasing table for you, the system ignores (skips) any DROP commands in the DDL file, and substitutes the value configured in the AutoMigrationTableName attribute for all occurrences of the word ACTIVE in the file.

Server Migration with Database Leasing on RAC Clusters

When using server migration with database leasing on RAC Clusters, Oracle recommends synchronizing all RAC nodes in the environment. If the nodes are not synchronized, it is possible that a Managed Server that is renewing a lease will evaluate that the value of the clock on the RAC node is greater than the timeout value of leasing table. If it is more than 30 seconds, the server instance will fail and restart with the following log message:

```
<Mar 29, 2013 2:39:09 PM EDT> <Error> <Cluster> <BEA-000150> <Server failed
to get a connection to the database in the past 60 seconds for lease renewal.
Server will shut itself down.>
```

For more information, see Configuring Time Synchronization for the Cluster in the *Installation and Upgrade Guide for Microsoft Windows*.

Non-Database Consensus Leasing

> **Note**
>
> Consensus leasing requires you to use Node Manager to control server instances within the cluster. Node Manager must be running on every machine hosting Managed Servers within the cluster, including any candidate machines for failed migratable servers. For more information, see Using Node Manager to Control Servers in *Administering Node Manager for Oracle WebLogic Server*.

In consensus leasing, there is no highly available database required. The cluster leader maintains the leases in-memory. All of the server instances renew their leases by contacting the cluster leader, however, the leasing table is replicated to other nodes of the cluster to provide failover.

The cluster leader is elected by all of the running server instances in the cluster. A server instance becomes a cluster leader only when it has received acceptance from the majority of the server instances. If Node Manager reports a server instance as shut down, the cluster leader assumes that server instance has accepted it as leader when counting the majority number of server instances.

Consensus leasing requires a majority of server instances to continue functioning. During network partition, the server instances in the majority partition will continue to run while those in the minority partition will voluntarily shut down, as they cannot contact the cluster leader or elect a new cluster leader since they will not have the majority of server instances. If the partition results in an equal division of server instances, then the partition that contains the cluster leader will survive while the other one will fail. Consensus leasing depends on the ability to contact Node Manager to receive the status of the server instances it is managing to count them as part of the majority of reachable server instances. If Node Manager cannot be contacted, due to loss of network connectivity or a hardware failure, the server instances it manages are not counted as part of the majority, even if they are running.

> **Note**
>
> If your cluster only contains two server instances, the cluster leader will be the majority partition if a network partition occurs. If the cluster leader fails, the surviving server instance will attempt to verify its status through Node Manager. If the surviving server instance is able to determine the status of the failed cluster leader, it assumes the role of cluster leader. If the surviving server instance cannot check the status of the cluster leader, due to machine failure or a network partition, it will voluntarily shut down as it cannot reliably determine if it is in the majority.
>
> To avoid this scenario, Oracle recommends to use a minimum of three server instances running on different machines.

If automatic server migration is enabled, server instances are required to contact the cluster leader and renew their leases periodically. Server instances will shut themselves down if they are unable to renew their leases. The failed server instances will automatically migrate to the machines in the majority partition.

Automatic Whole Server Migration

Learn the procedures for configuring automatic whole server migration and how whole server migration functions within a WebLogic Server environment.

Preparing for Automatic Whole Server Migration

Before configuring automatic whole server migration, be aware of the following requirements:

- To use whole server migration on Oracle Solaris Zones, you must use Exclusive-IP zones as described in Whole Server Migration with Solaris 10 Zones Configuration Guide v1.0 available in My Oracle Support.
- Each Managed Server uses the same subnet mask. Unicast and multicast communication among server instances requires each server instance to use the same subnet. Server migration will not work without configuring multicast or unicast communication.

  See Using IP Multicast and One-to-Many Communication Using Unicast.
- All server instances hosting migratable servers are time-synchronized. Although migration works when server instances are not time-synchronized, time-synchronized server instances are recommended in a clustered environment.
- If you are using different operating system versions among migratable servers, ensure that all versions support identical functionality for `ifconfig`.
- Automatic whole server migration requires Node Manager to be configured and running for IP migration and server restart on another machine.
- The primary interface names used by migratable servers are the same. If your environment requires different interface names, then configure a local version of `wlscontrol.sh` for each migratable server.

  For more information about `wlscontrol.sh`, see Using Node Manager to Control Servers in *Administering Node Manager for Oracle WebLogic Server*.
- For a list of supported database, see Databases Supporting WebLogic Server Features in *Oracle Fusion Middleware Supported System Configurations*.

- There is no built-in mechanism for transferring files that a server instance depends on between machines. Using a disk that is accessible from all machines is the preferred way to ensure file availability. If you cannot share disks between server instances, you must ensure that the contents of *domain_dir*`/bin` are copied to each machine.
- Ensure that the Node Manager security files are copied to each machine using the WLST command `nmEnroll()`. See Using Node Manager to Control Servers in *Administering Node Manager for Oracle WebLogic Server*.
- Use high availability storage for state data. For highest reliability, use a shared storage solution that is itself highly available, for example, a storage area network (SAN). See Using High Availability Storage for State Data.
- Use a central shared directory for persistent file store data. See Additional Requirement for High Availability File Stores in *Administering the WebLogic Persistent Store*.
- For capacity planning in a production environment, remember that server startup during migration taxes CPU utilization. You cannot assume that because a machine can handle a certain number of server instances running concurrently that it also can handle that same number of server instances starting up on the same machine at the same time.
- To support whole server migration on OCI, WebLogic requires the following, additional command-line tools be installed in order to assign virtual (or secondary) IP addresses to a compute instance's VNIC:
  - The OCI command-line interface (CLI). See Installing the CLI in the Oracle Cloud Infrastructure Documentation.
  - The jq command-line JSON processor. See the installation options on the jq download page.

Configuring Automatic Whole Server Migration

Before configuring server migration, ensure that your environment meets the requirements outlined in Preparing for Automatic Whole Server Migration.

To configure server migration for a Managed Server within a cluster, perform the following tasks:

1. Obtain floating IP addresses for each Managed Server that will have migration enabled.

   Each migratable server must be assigned a floating IP address which follows the server instance from one physical machine to another after migration. Any server instance that is assigned a floating IP address must also have `AutoMigrationEnabled` set to true.

   > **Note**
   >
   > The migratable IP address should not be present on the interface of any of the candidate machines before the migratable server is started.

2. Configure Node Manager. Node Manager must be running and configured to allow server migration.

   The Java version of Node Manager can be used for server migration on Windows or UNIX. The script-based version of Node Manager can be used for server migration on UNIX only.

   When using Java-based Node Manager, you must edit `nodemanager.properties` to add your environment `Interface` and `NetMask` values.

To determine the most appropriate `Interface` value for your environment, use the operating system utility to find the list of network interfaces available on the machine. On Unix platforms, this is typically the `ifconfig` command. On Windows platforms, this is typically the `ipconfig` command.

To determine `NetMask`, you can use the same `NetMask` value that may already be configured for addresses on that same interface to ensure that all traffic occurs on the same subnet. You can also specify a common `NetMask` value, and therefore specify a subnet for all WebLogic Server traffic.

The `nodemanager.properties` file is located in the directory specified in *`NodeManagerHome`*.

(UNIX) *`ORACLE_HOME`*`/user_projects/domains/`*`domain_name`*`/nodemanager`

(Windows) *`ORACLE_HOME`*`\oracle_common\common\nodemanager`

For information about `nodemanager.properties`, see Reviewing nodemanager.properties in *Administering Node Manager for Oracle WebLogic Server.*

If you are using the script-based version of Node Manager, edit `wlscontrol.sh` and set the `Interface` variable to the name of your network interface.

For general information on using Node Manager in server migration, see Node Manager Role in Whole Server Migration. See Node Manager Overview in *Administering Node Manager for Oracle WebLogic Server*.

3. If you are using a database to manage leasing information, configure the database for server migration according to the procedures outlined in High-availability Database Leasing. See Leasing.
4. If you are using database leasing within a test environment and you need to reset the leasing table, you should re-run the `leasing.ddl` script. This causes the correct tables to be dropped and re-created.
5. If you are using a database to store leasing information, set up and configure a data source according to the procedures outlined in High-availability Database Leasing.

   You should set `DataSourceForAutomaticMigration` to this data source in each cluster configuration.

   > **Note**
   >
   > XA data sources are not supported for server migration.

   For more information, see Configuring JDBC Data Sources in *Administering JDBC Data Sources for Oracle WebLogic Server*.
6. Grant superuser privileges to the `wlsifconfig.sh` script (on UNIX) or the `wlsifconfig.cmd` script (on Windows).

   This script is used to transfer IP addresses from one machine to another during migration. It must be able to run `ifconfig`, which is generally only available to superusers. You can edit the script so that it is invoked using `sudo`.

   Java-based Node Manager uses the `wlsifconfig.cmd` script, which uses the `netsh` utility.

   The `wlsifconfig` scripts are available in the *`WL_HOME`*`/common/bin` or *`DOMAIN_HOME`*`/bin/server_migration` directory.
7. Ensure that the following commands are included in your machine PATH:
   - `wlsifconfig.sh` (UNIX) or `wlsifconfig.cmd` (Windows)

- `wlscontrol.sh` (UNIX)
- `nodemanager.domains`

The `wlsifconfig.sh`, `wlsifconfig.cmd`, and `wlscontrol.sh` files are located in *`WL_HOME`*`/common/bin` or *`DOMAIN_HOME`*`/bin/server_migration`. The `nodemanager.domains` file is located in the directory specified in *`NodeManagerHome`*.

For Java-based Node Manager, *`NodeManagerHome`* is typically located in

- (UNIX) *`ORACLE_HOME`*`/user_projects/domains/`*`domain_name`*`/nodemanager`
- (Windows) *`ORACLE_HOME`*`\oracle_common\common\nodemanager`

For script-based Node Manager, this file's default *`NodeManagerHome`* location is *`WL_HOME`*`/common/nodemanager`, where *`WL_HOME`* is the location in which you installed WebLogic Server, for example, *`ORACLE_HOME`*`/wlserver`.

Depending on your default shell on UNIX, you may need to edit the first line of the `.sh` scripts.

8. This step applies only to the script-based version of Node Manager and UNIX. If you are using Windows, skip to step 9.

   The machines that host migratable servers must trust each other. For server migration to occur, it must be possible to get to a shell prompt using '`ssh/rsh machine_A`' from machine_B and vice versa without having to explicitly enter a username and password. Also, each machine must be able to connect to itself using SSH in the same way.

   > **Note**
   >
   > You should ensure that your login scripts (`.cshrc`, `.profile`, `.login`, and such) only echo messages from your shell profile if the shell is interactive. WebLogic Server uses an `ssh` command to login and echo the contents of the `server.state` file. Only the first line of this output is used to determine the server state.

9. Set the candidate machines for server migration. Each server instance can have a different set of candidate machines, or they can all have the same set.
10. Restart the Administration Server.

Using High Availability Storage for State Data

The server migration process migrates services, but not the state information associated with work in process at the time of failure.

To ensure high availability, it is critical that such state information remains available to the server instance and the services it hosts after migration. Otherwise, data about the work in process at the time of failure may be lost. State information maintained by a migratable server, such as the data contained in transaction logs, should be stored in a shared storage system that is accessible to any potential machine to which a failed migratable server might be migrated. For highest reliability, use a shared storage solution that is itself highly available (for example, a SAN). For more information, see Additional Requirement for High Availability File Stores in *Administering the WebLogic Persistent Store*.

In addition, if you are using a database to store leasing information, the *lease table*, described in the following sections, should also be stored in a high availability database. The lease table tracks the health and liveness of migratable servers. See Leasing.

Server Migration Processes and Communications

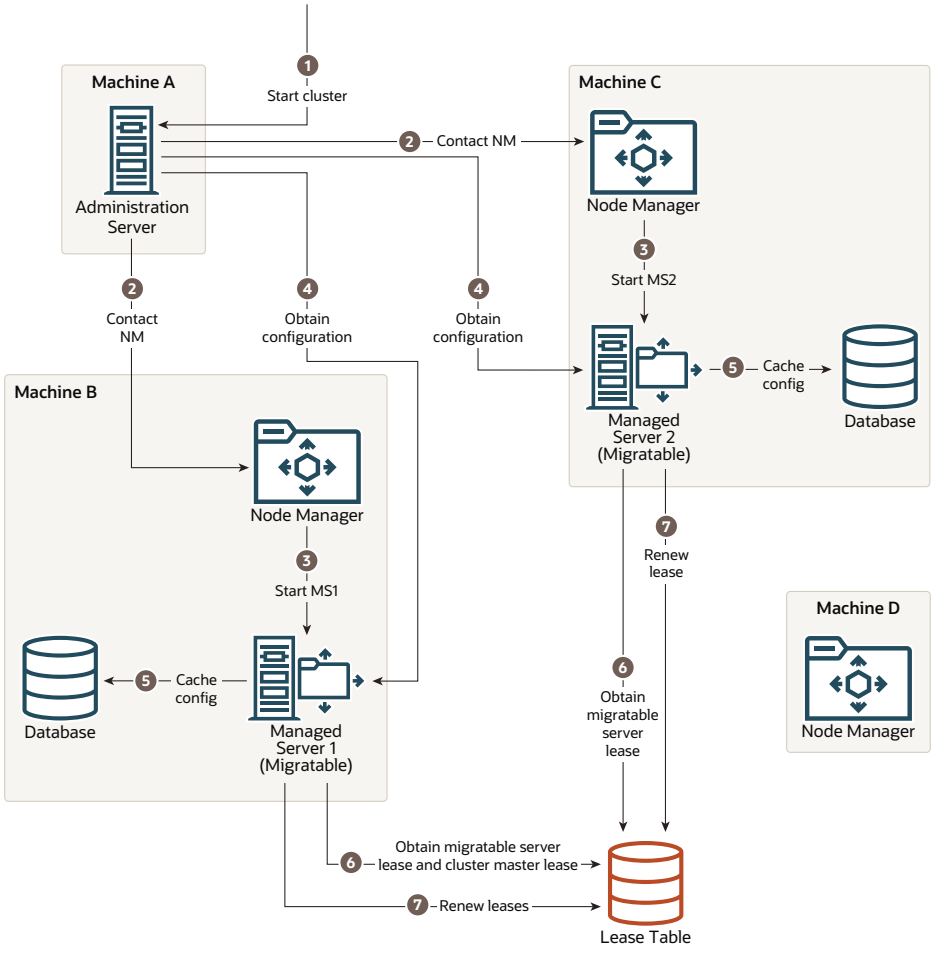
The following sections describe key processes in a cluster that contains migratable servers:

Startup Process in a Cluster with Migratable Servers

Figure 6-1 illustrates the process and communication that occurs during startup of a cluster that contains migratable servers.

The example cluster contains two Managed Servers, both of which are migratable. The Administration Server and the two Managed Servers each run on different machines. A fourth machine is available as a backup, if one of the migratable servers fails. Node Manager is running on the backup machine and on each machine with a running migratable server.

Figure 6-1 Startup of Cluster with Migratable Servers



The following key steps occur during startup of the cluster, as illustrated in Figure 6-1:

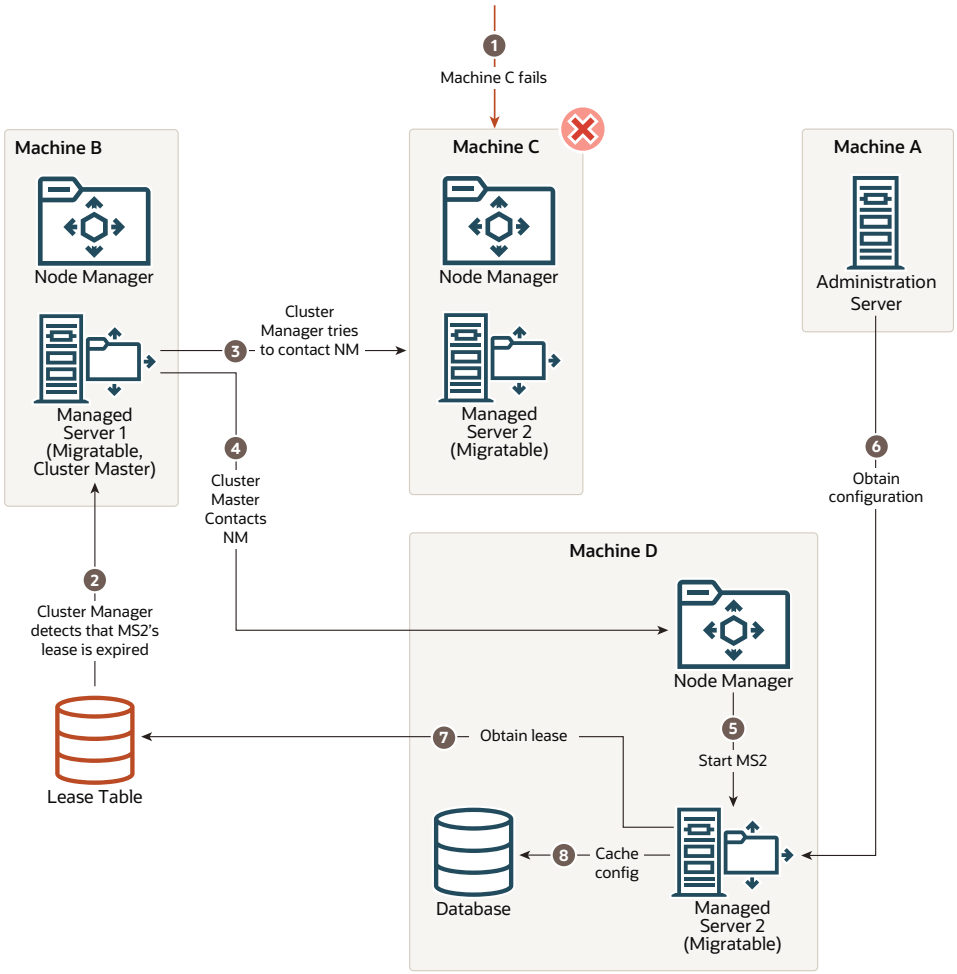
1. The administrator starts the cluster.
2. The Administration Server invokes Node Manager on Machines B and C to start Managed Servers 1 and 2, respectively. See Administration Server Role in Whole Server Migration.

3. The Node Manager instance on each machine starts the Managed Server that runs on that machine. See Node Manager Role in Whole Server Migration.
4. Managed Servers 1 and 2 contact the Administration Server for their configuration. See Migratable Server Behavior in a Cluster.
5. Managed Servers 1 and 2 cache the configuration with which they started.
6. Managed Servers 1 and 2 each obtain a migratable server lease in the lease table. Because Managed Server 1 starts first, it also obtains a cluster master lease. See Cluster Master Role in Whole Server Migration.
7. Managed Server 1 and 2 periodically renew their leases in the lease table, proving their health and liveness.

Automatic Whole Server Migration Process

Figure 6-2 illustrates the automatic migration process after the failure of the machine hosting Managed Server 2.

Figure 6-2 Automatic Migration of a Failed Server



1. Machine C, which hosts Managed Server 2, fails.
2. Upon its next periodic review of the lease table, the cluster master detects that Managed Server 2's lease has expired. See Cluster Master Role in Whole Server Migration.

3. The cluster master tries to contact Node Manager on Machine C to restart Managed Server 2, but it fails as Machine C is unreachable.

> **Note**
>
> If the Managed Server 2 lease had expired because it was hung, and Machine C was reachable, the cluster master would use Node Manager to restart Managed Server 2 on Machine C.

4. The cluster master contacts Node Manager on Machine D, which is configured as an available host for migratable servers in the cluster.
5. Node Manager on Machine D starts Managed Server 2. See Node Manager Role in Whole Server Migration.
6. Managed Server 2 starts and contacts the Administration Server to obtain its configuration.
7. Managed Server 2 caches the configuration with which it started.
8. Managed Server 2 obtains a migratable server lease.

During migration, the clients of the Managed Server that is migrating may experience a brief interruption in service; it may be necessary to reconnect. On Solaris and Linux operating systems, this can be done using the `ifconfig` command. The clients of a migrated server do not need to know the particular machine to which they have migrated.

When a migrated machine that previously hosted a server instance becomes available again, the migration process will be reversed (migrating the server instance back to its original host machine), which is known as *failback*. WebLogic Server does not automate the failback process. An administrator can accomplish failback by manually restoring the server instance to its original host.

The general procedures for restoring a server instance to its original host are as follows:

- Gracefully shut down the new instance of the server.
- After you have restarted the failed machine, restart Node Manager and the Managed Server.

The procedures you follow will depend on your server instance and network environment.

Administration Server Role in Whole Server Migration

In a cluster that contains migratable servers, the Administration Server:

- Invokes Node Manager on each machine that hosts cluster members to start the migratable servers. This is a prerequisite for server migratability, if a server instance was not initially started by Node Manager, it cannot be migrated.
- Invokes Node Manager on each machine involved in a manual migration process to stop and start the migratable server.
- Invokes Node Manager on each machine that hosts cluster members to stop server instances during a normal shutdown. This is a prerequisite for server migratability, if a server instance is shut down directly, without using Node Manager, when the cluster master detects that the server instance is not running, it will call Node Manager to restart it.

In addition, the Administration Server provides its regular domain management functionality, persisting configuration updates issued by an administrator, and providing a run-time view of the domain, including the migratable servers it contains.

Migratable Server Behavior in a Cluster

A migratable server is a clustered Managed Server that has been configured as migratable. A migratable server has the following key behaviors:

- If you are using a database to manage leasing information, during startup and restart by Node Manager, a migratable server adds a row to the lease table. The row for a migratable server contains a timestamp and the machine where it is running. See Leasing.
- When using a database to manage leasing information, a migratable server adds a row to the database as a result of startup. It tries to take on the role of cluster master and succeeds, if it is the first server instance to join the cluster.
- Periodically, the server renews its lease by updating the timestamp in the lease table.

  By default, a migratable server renews its lease every 30,000 milliseconds—the product of two configurable `ServerMBean` properties:

  – `HealthCheckIntervalMillis`, which by default is 10,000.

  – `HealthCheckPeriodsUntilFencing`, which by default is 3.
- If a migratable server fails to reach the lease table and renew its lease before the lease expires, it terminates as quickly as possible using a Java `System.exit`. In this case, the lease table still contains a row for that server instance. For information about how this relates to automatic migration, see Cluster Master Role in Whole Server Migration.
- During operation, a migratable server listens for heartbeats from the cluster master. When it detects that the cluster master is not sending heartbeats, it attempts to take over the role of cluster master and succeeds if no other server instance has claimed that role.

> **Note**
>
> During server migration, remember that server startup taxes CPU utilization. You cannot assume that because a machine can support a certain number of server instances running concurrently that they also can support that same number of server instances starting up on the same machine at the same time.

Node Manager Role in Whole Server Migration

The use of Node Manager is required for server migration. It must run on each machine that hosts or is intended to host.

Node Manager supports server migration in the following ways:

- Node Manager must be used for initial startup of migratable servers.

  You can also invoke Node Manager to start the server instance using the standalone Node Manager client; however, the Administration Server must be available so that the Managed Server can obtain its configuration.

> **Note**
>
> Migration of a server instance that is not initially started with Node Manager will fail.

- Node Manager must be used to suspend, shut down, or force shut down migratable servers.
- Node Manager tries to restart a migratable server whose lease has expired on the machine where it was running at the time of failure.

  Node Manager performs the steps in the server migration process by running customizable shell scripts that are provided with WebLogic Server. These scripts can start, restart and stop server instances, migrate IP addresses, and mount and unmount disks. The scripts are available for Solaris and Linux.

  – In an automatic migration, the cluster master invokes Node Manager to perform the migration.
  – In a manual migration, the Administration Server invokes Node Manager to perform the migration.

Cluster Master Role in Whole Server Migration

In a cluster that contains migratable servers, one server instance acts as the cluster master. Its role is to orchestrate the server migration process. Any server instance in the cluster can serve as the cluster master. When you start a cluster that contains migratable servers, the first server instance to join the cluster becomes the cluster master and starts the cluster manager service. If a cluster does not include at least one migratable server, it does not require a cluster master, and the cluster manager service does not start. In the absence of a cluster master, migratable servers can continue to operate, but server migration is not possible. The cluster master serves the following key functions:

- Issues periodic heartbeats to the other server instances in the cluster.
- Periodically reads the lease table to verify that each migratable server has a current lease. An expired lease indicates to the cluster master that the migratable server should be restarted.
- Upon determining that a migratable server's lease is expired, the cluster master waits for a period specified by the `FencingGracePeriodMillis` on the `ClusterMBean` and then tries to invoke the Node Manager process on the machine that hosts the migratable server whose lease is expired, in order to restart the migratable server.
- If unable to restart a migratable server whose lease has expired on its current machine, the cluster master selects a target machine in the following fashion:

  – If you have configured a list of preferred destination machines for the migratable server, the cluster master chooses a machine on that list, in the order the machines are listed.
  – Otherwise, the cluster master chooses a machine on the list of those configured as available for hosting migratable servers in the cluster.

    A list of machines that can host migratable servers can be configured at two levels: for the cluster as a whole and for an individual migratable server. You can define a machine list at both levels, but it is necessary to define a machine list on at least one level.
- To accomplish the migration of a server instance to a new machine, the cluster master invokes the Node Manager process on the target machine to create a process for the server instance.

  The time required to perform the migration depends on the server configuration and startup time.

- The maximum time taken for the cluster master to restart the migratable server is (`HealthCheckPeriodsUntilFencing` * `HealthCheckIntervalMillis`) + `FencingGracePeriodMillis`.
- The total time before the server instance becomes available for client requests depends on the server startup time and the application deployment time.

Whole Server Migration with Dynamic and Mixed Clusters

WebLogic Server supports whole server migration with dynamic and mixed clusters.

When a dynamic server in a dynamic cluster fails, the server instance will be migrated to a different physical machine upon failure as the same way a configured server in a configured or mixed cluster. While configuration differs depending on the cluster type, whole server migration behavior is the same for all clusters. See Dynamic Clusters.

Automatic whole server migration uses leasing to elect a cluster master, which is responsible for monitoring other cluster members and for restarting failed members hosted on other physical machines. You configure leasing in the cluster configuration. See Leasing.

Configuring Whole Server Migration with Dynamic Clusters

When configuring automatic whole server migration for configured clusters, select the individual server instances you want to migrate. You can also choose a subset of available machines to which you want to migrate server instances upon failure.

For a dynamic cluster, you can enable or disable automatic whole server migration in the server template. A dynamic cluster uses a single server template to define its configuration, and all dynamic server instances within the dynamic cluster inherit the template configuration. If you enable automatic whole server migration in the server template for a dynamic cluster, all dynamic server instances based on that server template are then enabled for automatic whole server migration. You cannot select individual dynamic server instances to migrate.

Also, you cannot choose the machines to which you want to migrate. After enabling automatic whole server migration in the server template for a dynamic cluster, all machines that are available to use for migration are automatically selected.

You cannot limit the list of candidate machines for migration that the dynamic cluster specifies, as the server template does not list candidate machines. The list of candidate machines for each dynamic server is calculated as follows:

```
ClusterMBean.CandidateMachinesForMigratableServers = { M1, M2, M3 }

dyn-server-1.CandidateMachines = { M1, M2, M3}
dyn-server-2.CandidateMachines = { M2, M3, M1 }
dyn-server-3.CandidateMachines = { M3, M1, M2 }
dyn-server-4.CandidateMachines = { M1, M2, M3 }
```

Configuring Whole Server Migration with Mixed Clusters

A mixed cluster contains both dynamic and configured servers. To enable automatic whole server migration for a mixed cluster:

1. Enable automatic whole server migration in the server template used by the dynamic server instances in the mixed cluster.

   All of the dynamic server instances based on that server template are then enabled for automatic whole server migration.

2. Manually enable automatic whole server migration for any of the configured server instances in the cluster and choose the machines to which you want to migrate if a server instance fails. For more information, see Automatic Whole Server Migration.

7

Service Migration

This chapter describes the service migration mechanisms supported by Oracle WebLogic Server.
This chapter focuses on migrating failed services. WebLogic Server also supports whole server-level migration, where a migratable server instance, and all of its services are migrated to a different physical machine upon failure. For information on failed server migration, see Whole Server Migration.

WebLogic Server also supports replication and failover at the application level. For more information, see Failover and Replication in a Cluster.

> **Note**
>
> Support for automatic whole server migration on Solaris 10 systems using the Solaris Zones feature can be found in Note 3: *Support For Sun Solaris 10 In Multi-Zone Operation* at http://www.oracle.com/technetwork/middleware/ias/oracleas-supported-virtualization-089265.html.

This chapter includes the following sections:

Understanding the Service Migration Framework

In a WebLogic Server cluster, most subsystem services are hosted homogeneously on all server instances in the cluster, enabling transparent failover from one server to another.

In contrast, *pinned services*, such as JMS-related services, the JTA Transaction Recovery Service, and user-defined singleton services are hosted on individual server instances within a cluster. For these services, the WebLogic Server migration framework supports failure recovery with *service migration*, as opposed to failover. See Migratable Services.

Service-level migration in WebLogic Server is the process of moving the pinned services from one server instance to a different available server instance within the cluster. Service migration is controlled by a logical *migratable target*, which serves as a grouping of services that is hosted on only one physical server instance in a cluster. You can select a migratable target in place of a server instance or cluster when targeting certain pinned services. High availability is achieved by migrating a migratable target from one clustered server instance to another when a problem occurs on the original server instance. You can also manually migrate a migratable target for scheduled maintenance, or you can configure the migratable target for automatic migration. See Understanding Migratable Targets In a Cluster.

The migration framework provides tools and infrastructure for configuring and migrating targets, and, in the case of automatic service migration, it leverages the WebLogic Server health monitoring subsystem to monitor the health of services hosted by a migratable target. See Migration Processing Tools and Automatic Service Migration Infrastructure. For definitions of the terms that apply to server and service migration, see Migration Terminology.

Migratable Services

WebLogic Server supports service-level migration for JMS-related services, the JTA Transaction Recovery Service, and user-defined singleton services. These are referred to as *migratable services* because you can move them from one server instance to another within a cluster. The following migratable services can be configured for automatic or manual migration.

JMS-related Services

> **Note**
>
> If you want to enable service migration on a cluster targeted JMS Server, SAF Agent, Path Service, or Custom Store, see Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*.

These are the migratable JMS services:

- JMS Server: Management containers for the queues and topics in JMS modules that are targeted to them. See JMS Server Configuration in *Administering JMS Resources for Oracle WebLogic Server*.
- Store-and-Forward (SAF) Service: Store-and-Forward messages between local sending and remote receiving endpoints, even when the remote endpoint is not available at the moment the messages are sent. Only sending SAF agents configured for JMS SAF (sending capability only) are migratable. See *Administering the Store-and-Forward Service for Oracle WebLogic Server*.
- Path Service: A persistent map that can be used to store the mapping of a group of messages in a JMS Message Unit-of-Order to a messaging resource in a cluster. It provides a way to enforce ordering by pinning messages to a member of a cluster hosting servlets, distributed queue members, or Store-and-Forward agents. One path service is configured per cluster. See Using the WebLogic Path Service in *Administering JMS Resources for Oracle WebLogic Server*.
- Custom Persistent Store: A user-defined, disk-based file store or JDBC-accessible database for storing subsystem data, such as persistent JMS messages or store-and-forward messages. See *Administering the WebLogic Persistent Store*.

Cluster targeted JMS services are distributed over the members of the clusters. Weblogic Server can be configured to automatically migrate cluster targeted services across cluster members dynamically. See Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*.

To ensure that singleton JMS services do not introduce a single point of failure for dependent applications in the cluster, WebLogic Server can be configured to automatically or manually migrate them to any server instance in the migratable target list. See Roadmap for Configuring Automatic Migration of JMS-related Services and Roadmap for Configuring Manual Migration of JMS-related Services.

JTA Transaction Recovery Service

The Transaction Recovery Service automatically attempts to recover transactions on system startup by parsing all transaction log records for incomplete transactions and completing them. See Transaction Recovery After a Server Fails in *Developing JTA Applications for Oracle WebLogic Server*.

User-defined Singleton Services

Within an application, you can define a singleton service that can be used to perform tasks that you want to be executed on only one member of a cluster at any give time. See Automatic Migration of User-Defined Singleton Services.

Understanding Migratable Targets In a Cluster

> **Note**
>
> If you want to enable service migration on a cluster targeted JMS Server, SAF Agent, Path Service, or Custom Store, see Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*.

You can also configure JMS and JTA services for high availability by using migratable targets. A migratable target is a special target that can migrate from one server instance in a cluster to another. As such, a migratable target provides a way to group migratable services that should move together. When the migratable target is migrated, all services hosted by that target are migrated.

A migratable target specifies a set of server instances that can host a target, and can optionally specify a user-preferred host for the services and an ordered list of candidate backup servers should the preferred server instance fail. Only one of these server instances can host the migratable target at a time.

Once you configure a service to use a migratable target, then it is independent from the server member that is currently hosting it. For example, if you configure a JMS server with a deployed JMS queue to use a migratable target, then the queue is independent when a specific server member is available. In other words, the queue is always available when the migratable target is hosted by any server instance in the cluster.

An administrator can manually migrate pinned migratable services from one server instance to another in the cluster, either in response to a server failure or as part of regularly scheduled maintenance. If you do not configure a migratable target in the cluster, migratable services can be migrated to any WebLogic Server instance in the cluster. See Roadmap for Configuring Manual Migration of JMS-related Services.

Policies for Manual and Automatic Service Migration

A migratable target provides migration policies that define whether the hosted services will be manually migrated (the system default) or automatically migrated from an unhealthy hosting server instance to a healthy active server instance with the help of the health monitoring subsystem. There is one type of manual service migration and two types of automatic service migration policies, as described in the following sections.

Manual Migration

When a migratable target uses the `manual` policy (the system default), an administrator can manually migrate pinned migratable services from one server instance to another in the cluster, either in response to a server failure or as part of regularly scheduled maintenance.

See Roadmap for Configuring Manual Migration of JMS-related Services.

Exactly-Once

This policy indicates that if at least one Managed Server in the candidate list is running, then the service will be active somewhere in the cluster if server instances fail or shut down (either gracefully or forcibly). It is important to note that this value can lead to target grouping. For example, if you have five `exactly-once` migratable targets and only start one Managed Server in the cluster, then all five targets will be activated on that server instance.

> **Note**
>
> As a best practice, a migratable target hosting a path service should always be set to `exactly-once`, so if its hosting server member fails or shut down, the path service will automatically migrate to another server instance and will always be active in the cluster.

Example use-case for JMS servers:

A domain has a cluster of three Managed Servers, with one JMS server deployed on a member server in the cluster. Applications deployed to the cluster send messages to the queues targeted to the JMS server. MDBs in another domain drain the queues associated with the JMS server. The MDBs only want to drain from one set of queues, not from many instances of the same queue. In other words, this environment uses clustering for scalability, load balancing, and failover for its applications, but not for its JMS server. Therefore, this environment would benefit from the automatic migration of the JMS server as an `exactly-once` service to an available cluster member.

See Roadmap for Configuring Automatic Migration of JMS-related Services.

Failure-Recovery

This policy indicates that the service will only start if its user-preferred server (UPS) is started. If an administrator manually shuts down the UPS, either gracefully or forcibly, then a `failure-recovery` service will not migrate. However, if the UPS fails due to an internal error, then a `failure-recovery` service will be migrated to another candidate server instance. If such a candidate server instance is unavailable (due to a manual shutdown or an internal failure), then the migration framework will first attempt to reactivate the service on its UPS server. If the UPS server is not available at that time, then the service will be migrated to another candidate server instance.

For migration of cluster targeted JMS services, you can configure a Store with failure recovery parameters.

Example use-case for JMS servers:

A domain has a cluster of three Managed Servers, with a JMS server on each member server and a distributed queue member on each JMS server. There is also an MDB targeted to the cluster that drains from the distributed queue member on the local server member. In other words, this environment uses clustering for overall scalability, load balancing, and failover. Therefore, this environment would benefit from the automatic migration of a JMS server as an `failure-recovery` service to a UPS member.

> **Note**
>
> If a server instance is also configured to use the automatic whole server migration framework, which will shut down the server when its expired lease cannot be renewed, then any `failure-recovery` services configured on that server instance will not automatically migrate, no matter how the server instance is manually shut down by an administrator (for example, force shutdown versus graceful shutdown). See Automatic Whole Server Migration.

See the Roadmap for Configuring Automatic Migration of JMS-related Services.

Options for Attempting to Restart Failed Services Before Migrating

A migratable target provides options to attempt to deactivate and reactivate a failed service, instead of migrating the service. See In-Place Restarting of Failed Migratable Services.

For more information about the default values for all migratable target options, see `MigratableTargetMBean` in the *MBean Reference for Oracle WebLogic Server*.

User-Preferred Servers and Candidate Servers

When deploying a JMS service to the migratable target, you can select the UPS target to host the service. When configuring a migratable target, you can also specify constrained candidate servers (CCS) that can potentially host the service should the user-preferred server fail. If the migratable target does not specify a CCS, the JMS server can be migrated to any available server instance in the cluster.

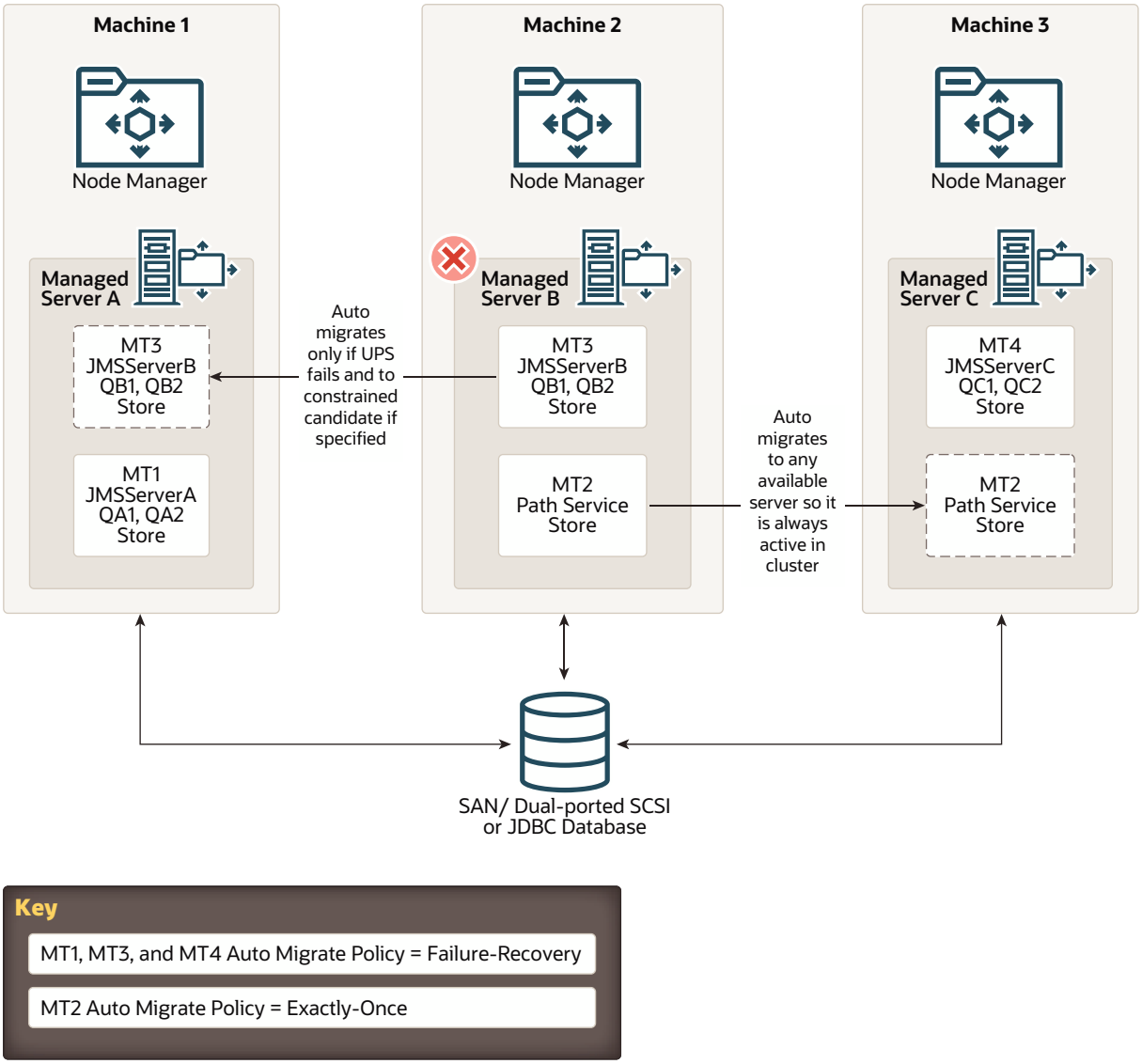
WebLogic Server enables you to create separate migratable targets for JMS services. This allows you to always keep each service running on a different server instance in the cluster, if necessary. Conversely, you can configure the same selection of server instances as the constrained candidate servers for both JTA and JMS, to ensure that the services remain co-located on the same server instance in the cluster.

Example Migratable Targets In a Cluster

Figure 7-1 shows a cluster of three Managed Servers, all hosting migratable targets. Server A is hosting a migratable target (MT1) for JMS server A (with two queues) and a custom store; Server B is hosting MT2 for a path service and a custom store and is also hosting MT3 for JMS server B (with two queues) and a custom store; Server C is hosting MT4 for JMS Server C (with two queues) and a custom store.

All the migratable targets are configured to migrate automatically with the MT1, MT3, and MT4 targets using the `failure-recovery` policy, and the MT2 target using the `exactly-once` policy.

Figure 7-1 Migratable Targets In a Cluster



In the above example, the MT2 `exactly-once` target will automatically start the path service and store on any running Managed Server in the candidate list. This way, if the hosting server should fail, it guarantees that the services will always be active somewhere in the cluster, even if the target's UPS is shut down gracefully. However, as described in Policies for Manual and Automatic Service Migration, this policy can also lead to target grouping with multiple JMS services being hosted on a single server instance.

Whereas, if the UPS is shut down gracefully or forcibly, then the MT1, MT3, and MT4 `failure-recovery` targets will automatically start the JMS server and store services on its UPS, but the pinned services will not migrate anywhere. However, if the UPS shuts down due to an internal error, then the services will be migrated to another candidate server.

Targeting Rules for JMS Servers

When you do not use migratable targets, a JMS server can be targeted to a dynamic cluster or to a specific cluster member and can use a custom store. However, when targeted to a migratable target, a JMS server must use a custom persistent store, and must be targeted to

the same migratable target used by the custom store. A JMS server, SAF agent, and custom store can share a migratable target. See Custom Store Availability for JMS Services.

If the JMS system resource target is cluster, WebLogic Server will create the migratable targets for each server instance in a cluster and then create separate JMS servers that are targeted individually to each migratable target.

Targeting Rules for SAF Agents

When you do not use migratable targets, a SAF agent can be targeted to an entire cluster or a list of multiple server instances in a cluster, with the requirement that the SAF agent and each server instance in the cluster must use the default persistent store. However, when targeted to a migratable target, a SAF agent must use a custom persistent store, and must be targeted to the same migratable target used by the custom store, similar to a JMS server. A SAF agent, JMS server, and custom store can share a migratable target. See Special Considerations When Targeting SAF Agents or Path Service.

WebLogic Server will create the migratable targets for each server in a cluster and then create separate SAF agents that are targeted individually to each migratable target. This handling increases throughput and high availability.

In addition, consider the following topics when targeting SAF agents to migratable targets.

Re-targeting SAF Agents to Migratable Targets

To preserve SAF message consistency, WebLogic Server prevents you from retargeting an existing SAF agent to a migratable target. Instead, you must delete the existing SAF agent and configure a new SAF agent with the same values and target it to a migratable target.

Targeting Migratable SAF Agents For Increased Message Throughput

When you do not use migratable targets, a SAF agent can be targeted to an entire cluster or multiple server instances in a cluster for increased message throughput. However, when a SAF agent is targeted to a migratable target, it cannot be targeted to any other server instances in the cluster, including an entire cluster. Therefore, if you want to increase throughput by importing a JMS destination to multiple SAF agents on separate server instances in a cluster, then you should create migratable targets for each server instance in the cluster and then create separate SAF agents that are targeted individually to each migratable target.

Targeting SAF Agents For Consistent Quality-of-Service

A WebLogic Server administrator has the freedom to configure and deploy multiple SAF agents in the same cluster or on the same server instance. As such, there could be situations where the same server instance has both migratable SAF agents and non-migratable SAF agents. For such cases, the behavior of a JMS client application may vary depending on which SAF agent handles the messages.

For example, an imported destination can be deployed to multiple SAF agents, and messages sent to the imported destination will be load-balanced among all SAF agents. If the list of the SAF agents contains non-migratable agents, the JMS client application may have a limited sense of high availability. Therefore, a recommended best practice is to deploy an imported destination to one or more SAF agents that provide the same level of high availability functionality. In other words, to ensure consistent forwarding quality and behavior, you should target the imported destination to a set of SAF agents that are all targeted to migratable targets or to non-migratable targets.

Targeting Rules for Path Service

When you do not use migratable targets, a path service is targeted to a single member of a cluster and can use either the default file store or a custom store. However, when targeted to a migratable target, a path service cannot use the default store, so a custom store must be configured and targeted to the same migratable target. As an additional best practice, the path service and its custom store should be the only users of that migratable target. Whereas, a JMS server, SAF agent, and custom store can share a migratable target.

Special Considerations For Targeting a Path Service

As a best practice, when the path service for a cluster is targeted to a migratable target, the path service and its custom store should be the only users of that migratable target.

When a path service is targeted to a migratable target, its provides enhanced storage of message unit-of-order (UOO) information for JMS distributed destinations, since the UOO information will be based on the entire migratable target instead of being based only on the server instance hosting the distributed destinations member.

Targeting Rules for Custom Stores

All JMS-related services require a custom persistent store that is targeted to the same migratable targets as the JMS services. When a custom store is targeted to a migratable target, the store `<directory>` parameter must be configured so that the store directory is accessible from all candidate server members in the migratable target.

If the JMS system resource target is the cluster, WebLogic Server will create the migratable targets for each server instance in a cluster and then create separate JMS servers and file stores that are targeted individually to each migratable target.

For more information, see Custom Store Availability for JMS Services.

Migratable Targets For the JTA Transaction Recovery Service

For JTA, a migratable target configuration should not be configured because a migratable target is automatically defined for JTA at the server level. In the Remote Console, go to **Environment**: **Servers**: *myServer*, and then **JTA Migratable Target**. To enable JTA automatic migration, on the **Migration** page, select the desired **JTA Migration Policy** . The default migration policy for JTA is **manual**. For automatic service migration, select either **failure-recovery** or **shutdown-recovery**. This means that the Transaction Recovery Service will only start if its UPS is started. If an administrator shuts down the UPS, either gracefully or forcibly, this service will not be migrated.

However, if the UPS shuts down due to an internal error, then this service will be migrated to another candidate server instance. If such a candidate server instance is unavailable (due to a manual shutdown or an internal failure), then the migration framework will first attempt to reactivate the service on its UPS server instance. If the UPS server is not available at that time, then the service will be migrated to another candidate server instance.

Migration Processing Tools

WebLogic Server migration framework provides infrastructure and facilities to perform the manual or automatic migration of JMS-related services and the JTA Transaction Recovery Service. By default, an administrator must manually execute the process to successfully

migrate the services from one server instance to another server instance. However, these services can also be easily configured to automatically migrate in response to a server failure.

WebLogic Scripting Tool

An administrator can use the WLST command-line interface utility to manage the life cycle of a server instance, including configuring and performing the migration process.

For more information, see Life Cycle Commands in *WLST Command Reference for WebLogic Server*.

Automatic Service Migration Infrastructure

The service migration framework depends on the following components to monitor server health issues and automatically migrate the pinned services to a healthy server instance.

Leasing for Migratable Services

Leasing is the process WebLogic Server uses to manage services that are required to run on only one member of a cluster at a time. Leasing ensures exclusive ownership of a cluster-wide entity. Within a cluster, there is a single owner of a lease. Additionally, leases can failover in case of server or cluster failure. This helps to avoid having a single point of failure. For more information, see Leasing.

The `Automatic Migration` option requires setting the cluster `Migration Basis` policy to either `Database` or `Consensus` leasing.

Database Leasing

If you are using a high availability database, such as Oracle RAC, to manage leasing information, configure the database for server migration according to the procedures outlined in High Availability Database Leasing.

Consensus Leasing

Setting `Migration Basis` to `Consensus` leasing means that the member servers maintain leasing information in-memory, which removes the requirement of having a high availability database to use leasing. This version of leasing requires that you use Node Manager to control server instances within the cluster. It also requires that all server instances that are migratable, or which could host a migratable target, have a Node Manager instance associated with them. Node Manager is required for health monitoring information about the member server instances involved. For more information, see Non-database Consensus Leasing.

> **Note**
>
> As a best practice, Oracle recommends configuring database leasing instead of consensus leasing.

Node Manager

When you use automatic service migration, Node Manager is required for health monitoring information about the member servers, as follows:

- Consensus leasing: Node Manager must be running on every machine hosting Managed Servers within the cluster.
- Database leasing: Node Manager must be running on every machine hosting Managed Servers within the cluster only if pre/post-migration scripts are defined. If pre/post-migration scripts are not defined, then Node Manager is not required.

For general information about configuring Node Manager, see Using Node Manager to Control Servers in *Administering Node Manager for Oracle WebLogic Server*.

Administration Server Not Required When Migrating Services

To eliminate a single point of failure during migration, automatic service migration of migratable services is *not* dependent on the availability of the Administration Server at the time of migration.

Service Health Monitoring

To accommodate service migration requests, the migratable target performs basic health monitoring on migratable services that are deployed on it that implement a health monitoring interface. The advantage of having a migratable target perform this job is that it is guaranteed to be local. In addition, the migratable target has a direct communication channel to the leasing system and can request to release the lease (thus triggering a migration) when bad health is detected.

How Health Monitoring of the JTA Transaction Recovery Service Triggers Automatic Migration

When JTA has automatic migration enabled, the server defaults to shutting down if the JTA subsystem reports itself as unhealthy (`FAILED`). For example, if any I/O error occurs when accessing the transaction log, then JTA health state will change to `FAILED`.

When the primary server fails, the migratable service framework automatically migrates the Transaction Recovery Service to a backup server. The automatic service migration framework selects a backup server from the configured candidate servers. If a backup server fails before completing the transaction recovery actions and restarted, then the Transaction Recovery Service will eventually be migrated to another server instance in the cluster (either the primary server will reclaim it or the migration framework will notice that the backup server instance's lease has expired).

After successful migration, if the backup server is shut down normally and rebooted, then the Transaction Recovery Service will again be activated on the backup server. This is consistent with manual service migration. As with manual service migration, the Transaction Recovery Service cannot be migrated from a running primary server.

How Health Monitoring of JMS-related Services Triggers Automatic Migration

When the JMS-related services have automatic migration enabled:

- **JMS Server**: Maintains its run-time health state and registers and updates its health to the health monitoring subsystem. When a service the JMS server depends upon, such as its targeted persistent store, reports the `FAILED` health state, it is detected by the migration framework. The migration process takes place based on the migratable target's configured automatic migration policy. Typically, the migration framework deactivates the JMS server and other users of the migratable target on the current *user-preferred* server and migrates them onto a healthy available server instance from the constrained candidate server list.
- **SAF Service**: The health state of the SAF service comes from its configured SAF agents. If the SAF service detects an unhealthy state, the whole SAF agent instance will be

reported as unhealthy. The SAF agent has the same health monitoring capabilities as a JMS server. Typically, the migration framework deactivates the SAF agent on the current user-preferred server instance and migrates it onto a healthy available server instance from the constrained candidate server list.

- **Path Service**: The path service itself will not change its health state, but instead depends on the server instance and its custom store to trigger migration.
- **Persistent Store**: Registers its health to the health monitoring subsystem. If there are any errors reported by the I/O layer, such that if the persistent store cannot continue with read/write and needs to be restarted before it can guarantee data consistency then the store's health is marked as `FAILED` and reported as `FAILED` to the health monitoring subsystem. This is detected by the automatic migration framework and triggers the auto-migration of the store and the subsystem services that are depending on that store from the current user-preferred server instance onto a healthy available server instance from the constrained candidate server list.

In-Place Restarting of Failed Migratable Services

Migratable services, such as JMS, have a unique feature of restarting the failed services automatically and can recover from temporary store failure without restarting the whole server. It is sometimes beneficial for the service to be restarted in place, instead of migrated. Therefore, migratable targets provide **Restart In Place** option to attempt to deactivate and reactivate a failed service, instead of migrating the service. A custom store targeted to dynamic cluster also provides **Restart In Place** configuration option for cluster targeted JMS services.

The migration framework only attempts to restart a service if the server health is satisfactory (for example, in a `RUNNING` state). If the server instance is not healthy, the framework immediately proceeds to the migration stage, skipping all in-place restarts.

The cluster Singleton Monitor checks for the `RestartOnFailure` value in the service `MigratableTargetMBean`. If the value is `false`, the service is migrated automatically. If the value is `true`, the migration framework attempts to deactivate and activate in place. If the reactivation fails, the migration framework pauses for the user-specified `SecondsBetweenRestarts` seconds. The process is repeated for the specified `NumberOfRestartAttempts` attempts. If all restart attempts fail, the service is migrated to a healthy server member.

For more information, see Restart In Place in *Administering the WebLogic Persistent Store*.

Migrating a Service From an Unavailable Server

There are special considerations when you migrate a service from a server instance that has crashed or is unavailable to the Administration Server. If the Administration Server cannot reach the previously active host of the service when you perform the migration, the Managed Server's local configuration information (for example, migratable target) will not be updated to reflect that it is no longer the active host for the service. In this situation, you must purge the unreachable Managed Server's local configuration cache before restarting it. This prevents the previous active host from hosting a service that has been migrated to another Managed Server.

JMS and JTA Automatic Service Migration Interaction

In some automatic service migration cases, the migratable targets for JMS services and the JTA Transaction Recovery Service can be migrated to different candidate servers with uncommitted transactions in progress. However, JMS and JTA service states are independent

in time and location; therefore, JMS service availability does not depend on JTA transaction recovery being complete.

However, *in-doubt* transactions will not resolve until both services are running and can re-establish communication. An in-doubt transaction is an incomplete transaction that involves multiple participating resources (such as a JMS server and a database), where one or more of the resources are waiting for the transaction manager to tell them whether to rollback, commit, or forget their part of the transaction. Transactions can become in-doubt if they are in-progress when a transaction manager or participating resource crashes.

JTA continues to attempt to recover transactions when a resource is not available until the recovery abandon time period expires, which defaults to 24 hours.

Pre-Migration Requirements

WebLogic Server imposes certain constraints and prerequisites for service configuration to support service migration.

These constraints are service-specific and depend on your enterprise application architecture.

Custom Store Availability for JMS Services

Migratable JMS-related services cannot use the default persistent store, so you must configure a custom store and target it to the same migratable target as the JMS server or SAF agent. As a best practice, a path service should use its own custom store and migratable target.

The custom file store or JDBC store must either be:

- Accessible from all candidate server members in the migratable target.
  - If the application uses file-based persistence (file store), the store's `<directory>` parameter must be configured so that it is accessible from all candidate server members in the migratable target. For highest reliability, use a shared storage solution that is itself highly available (for example, a storage area network (SAN) or a dual-ported SCSI disk).
  - If the application uses JDBC-based persistence (JDBC store), then the JDBC connection information for that database instance, such as data source and connection pool, has to be available from all candidate servers members.
- Migrated to a backup server target by pre-migration and post-migration scripts in the `ORACLE_HOME/user_projects/domains/`*`mydomain`*`/bin/service_migration` directory, where *`mydomain`* is a domain-specific directory, with the same name as the domain.

> **Note**
>
> Basic directions for creating pre-migration and post-migration scripts are provided in the `readme.txt` file in this directory.

In some cases, scripts may be needed to dismount the disk from the previous server and mount it on the backup server. These scripts are configured on Node Manager, using the `PreScript()` and `PostScript()` methods in the `MigratableTargetMBean` in the *MBean Reference for Oracle WebLogic Server*. In other cases, a script may be needed to move (not copy) a custom file store directory to the backup server instance. Do not leave the old configured file store directory for the next time if the migratable target is to host the old

server instance. Therefore, the WebLogic Server administrator should delete or move the files to another directory.

Default File Store Availability for JTA

To migrate the JTA Transaction Recovery Service from a failed server instance in a cluster to another server instance (the backup server instance) in the same cluster, the backup server instance must have access to the transaction log (TLOG) records from the failed server. TLOG records are stored in the default persistent store for the server.

If you plan to use service migration in the event of a failure, you must configure the default persistent store so that it stores records in a shared storage system that is accessible to any potential machine to which a failed migratable server might be migrated. For highest reliability, use a shared storage solution that is itself highly available—for example, a SAN or a dual-ported disk. In addition, only JTA and other non-migratable services can share the same default store.

Optionally, you may also want to use pre-migration and post-migration scripts to perform any unmounting and mounting of shared storage, as needed. Basic directions for creating pre-migration and post-migration scripts are provided in a `readme.txt` file in the *ORACLE_HOME*`/user_projects/domains/`*mydomain*`/bin/service_migration` directory, where `mydomain` is a domain-specific directory, with the same name as the domain.

Server State and Manual Service Migration

For automatic migration, when the current (source) server fails, the migration framework will automatically migrate the Transaction Recovery Service to a target backup server.

For manual migration, you cannot migrate the Transaction Recovery Service to a backup server instance from a running server instance. You must stop the server instance before migrating the Transactions Recovery Service.

Table 7-1 shows the support for migration based on the running state.

Table 7-1 Server Running State and Manual Migration Support

Server State Information for Current Server	Server State Information for Backup Server	Messaging Migration Allowed?	JTA Migration Allowed?
Running	Running	Yes	No
Running	Standby	Yes	No
Running	Not running	Yes	No
Standby	Running	Yes	No
Standby	Standby	Yes	No
Standby	Not Running	Yes	No
Not Running	Running	Yes	Yes
Not Running	Standby	Yes	No
Not Running	Not Running	Yes	Yes

Roadmap for Configuring Automatic Migration of JMS-Related Services

For automatic migration, the WebLogic administrator can specify a migratable target for JMS-related services, such as JMS servers and SAF agents. The administrator can also configure migratable services that will be automatically migrated from a failed server based on WebLogic Server health monitoring capabilities.

> **Note**
>
> If you want to enable service migration on a cluster targeted JMS Server, SAF Agent, Path Service, or Custom Store, see Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*.
>
> JMS services can be migrated independently of the JTA Transaction Recovery Service. However, since the JTA Transaction Recovery Service provides the transaction control of the other subsystem services, it is usually migrated along with the other subsystem services. This ensures that the transaction integrity is maintained before and after the migration of the subsystem services.

A dynamic or mixed cluster allows simplified configuration for automatic migration of JMS-related services. For more information, see Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*.

To configure automatic JMS service migration on a migratable target within a cluster, perform the following tasks:

Step 1: Configure Managed Servers and Node Manager

Configure the Managed Servers in the cluster for migration, including assigning Managed Servers to a machine. In certain cases, Node Manager must also be running and configured to allow automatic server migration.

For step-by-step instructions for using the WebLogic Remote Console to complete these tasks, see the following topics in the *Oracle WebLogic Remote Console Online Help*:

- Create Managed Servers

  > **Note**
  >
  > It is required to configure a unique `Listen Address` for each Managed Server instance that will host migratable JMS services. Otherwise, migration may fail.

- Create and Configure Machines

> **Note**
>
> In general, Machines are required to enable Node Managers and the WebLogic Remote Console to start and stop WebLogic Server instances. When not using a Node Manager, Machines generally are not needed.

- Configure Node Manger, see Step 7. in Configure Machines

> **Note**
>
> For automatic service migration, Consensus leasing requires that you use Node Manager to control server instances within the cluster and that all migratable servers must have a Node Manager instance associated with them. For Database leasing, Node Manager is required only if pre-migration and post-migration scripts are defined. If pre-migration and post-migration scripts are not defined, then Node Manager is not required. For more information, see Database Leasing and Consensus Leasing.

For general information on configuring Node Manager, see Using Node Manager to Control Servers in *Administering Node Manager for Oracle WebLogic Server*.

Step 2: Configure the Migration Leasing Basis

In the Remote Console, on the **Environment**: **Clusters**: *myCluster*: **Migration** page, configure the cluster **Migration Basis** according to how your data persistence environment is configured, selecting either **database** or **consensus**. See Leasing for Migratable Services.

Database Leasing requires that the **Data Source For Automatic Migration** field is set with a reference to valid JDBC system resource. It also requires that a leasing table be created on that resource. See High Availability Database Leasing.

> **Note**
>
> As a best practice, Oracle recommends configuring database leasing instead of consensus leasing.

Step 3: Configure Migratable Targets

You should perform this step before targeting any JMS-related services or enabling the JTA Transaction Recovery Service migration.

Configuring a Migratable Server as an Automatically Migratable Target

In the Remote Console, the **Migratable Target** table displays the migratable targets of *myServer* (migratable), which are automatically generated for each running server instance in a cluster. However, these are only generic templates and still need to be targeted and configured for automatic migration.

Create a New Migratable Target

When creating a new migratable target, the WebLogic Remote Console provides a mechanism for creating, targeting, and selecting a migration policy.

Select a User Preferred Server

When you create a new migratable target using the WebLogic Remote Console, you can initially choose a preferred server instance in the cluster on which to associate the target. In the **Edit Tree**, on the **Environment**: **Migratable Targets**: *myMigratableTarget* page, select the **User Preferred Server**, which is the most appropriate server instance for hosting the migratable target.

> **Note**
>
> An automatically migrated service may not end up being hosted on the specified **User Preferred Server**. To verify which server is hosting a migrated service, use the WebLogic Remote Console to check the **Current Hosting Server** information on the *myMigratableTarget* page.

Select a Service Migration Policy

The default migration policy for migratable targets is **manual**, so you must select one of the following auto-migration policies:

- **Auto-Migrate Exactly-Once Services**: It indicates that if at least one Managed Server in the candidate list is running, then the service will be active somewhere in the cluster if server instances should fail or are shut down (either gracefully or forcibly).

  > **Note**
  >
  > This value can lead to target grouping. For example, if you have five **exactly-once** migratable targets and only start one Managed Server in the cluster, then all five targets will be activated on that server instance.

- **Auto-Migrate Failure-Recovery Services**: This policy indicates that the service will only start if its UPS is started. If an administrator shuts down the UPS either gracefully or forcibly, this service will not be migrated. However, if the UPS fails due to an internal error, the service will be migrated to another candidate server instance. If such a candidate server instance is unavailable (due to a manual shutdown or an internal failure), then the migration framework will first attempt to reactivate the service on its UPS server. If the UPS server is not available at that time, then the service will be migrated to another candidate server instance.

For more information, see Policies for Manual and Automatic Service Migration.

Optionally Select Constrained Candidate Servers

When creating migratable targets that use the **exactly-once** services migration policy, you may also want to restrict the potential member servers to which JMS servers can be migrated. A recommended best practice is to limit each migratable target's candidate server set to a primary, secondary, and perhaps a tertiary server instance. Then as each server starts, the

migratable targets will be restricted to their candidate server instances, rather than being satisfied by the first server instance to come online. Administrators can then manually migrate services to idle server instances.

For the cluster's path service, however, the candidate server instances for the migratable target should be the entire cluster, which is the default setting.

In the WebLogic Remote Console, on the **Migratable Targets**: *myManagedServer* **(migratable)**: **Candidates** page, the **Constrained Candidate Servers Available** selector lists all of the Managed Server instances that could possibly support the migratable target. The Managed Servers become valid Candidate Servers when you move them into the **Chosen** list.

Optionally Specify Pre/Post-Migration Scripts

After creating a migratable target, you may also want to specify whether you are providing any pre-migration and post-migration scripts to perform any unmounting and mounting of the shared custom file store, as needed.

- **Pre-Migration Script Path**: It is the path to the pre-migration script to run before a migratable target is actually activated.
- **Post-Migration Script Path**: It is the path to the post-migration script to run after a migratable target is fully deactivated.
- **Post-Migration Script Failure Cancels Automatic Migration**: It specifies whether or not a failure during execution of the post-deactivation script is fatal to the migration.
- **Allow Post-Migration Script To Run On a Different Machine**: It specifies whether or not the post-deactivation script is allowed to run on a different machine.

The pre-migration and post-migration scripts must be located in the *`ORACLE_HOME`*`/user_projects/domains/`*`mydomain`*`/bin/service_migration` directory, where *mydomain* is a domain-specific directory, with the same name as the domain. For your convenience, sample pre-migration and post-migration scripts are provided in this directory.

Optionally Specify In-Place Restart Options

Migratable targets provide `restart-in-place` options to attempt to deactivate and reactivate a failed service, instead of migrating the service. See In-Place Restarting of Failed Migratable Services.

Step 4: Configure and Target Custom Stores

As per Custom Store Availability for JMS Services, JMS-related services require you to configure a custom persistent store that is targeted to a dynamic cluster or to the same migratable targets as the JMS services. Ensure that the store is either:

- Configured such that all the candidate server instances in a migratable target have access to the custom store.
- Migrated by pre-migration and post-migration scripts. For more information, see Optionally Specify Pre/Post-Migration Scripts.

Step 5: Configure Destinations and Connection Factories

You can optionally configure Destinations and Connection Factories for your JMS Servers and SAF Agents using system modules. Remember the following best practice:

- Use JMS system modules rather than deployment modules. The WebLogic Remote Console only provides the ability to configure system modules. For more information, see JMS System Module Configuration in *Administering JMS Resources for Oracle WebLogic Server*.
- Create one system module per anticipated target set, and target the module to a single cluster. For example, if you plan to have one destination that spans a single JMS server and another destination that spans six JMS servers, create two modules and target both of them to the same cluster.
- Configure one subdeployment per module and populate the subdeployment with a homogeneous set of either JMS server of JMS SAF agent targets. Do not include WebLogic Server or cluster names in the subdeployment.
- Target connection factories to clusters for applications running on the same cluster. You can use default targeting to inherit the module target.
  - Custom connection factories are used to control client behavior, such as load balancing. They are targeted just like any other resource, but in the case of a connection factory, the target set has a special meaning.
  - You can target a connection factory to a cluster, WebLogic Server, or to a JMS server or SAF agent (using a subdeployment). There is a performance advantage to targeting connection factories to the exact JMS servers or SAF agents that the client will use, as the target set for a connection factory determines the candidate set of host server instances for a client connection.
  - Targeting to the exact JMS servers or SAF agents reduces the probability of client connections that connect to server instances which do not have a JMS server or SAF agent in cases where there is not a SAF agent on every clustered server instance. If no JMS server or SAF agent exists on a connection host, the client request must always double-hop the route from the client to the connection host server, then ultimately on to the JMS server or SAF agent.
- For other JMS module resources, such as destinations, target using a subdeployment. Do not use default targeting.
- As you add or remove JMS servers or SAF agents, remember to also add or remove JMS servers or SAF agents to your module subdeployment(s).
- For more information, see Best Practices for JMS Beginners and Advanced Users in *Administering JMS Resources for Oracle WebLogic Server*.

Step 6: Target the JMS Services

When using migratable targets, you must target your JMS service to the same migratable target used by the custom persistent store. In the event that no custom store is specified for a JMS service that uses a migratable target, then a validation message will be generated, followed by failed JMS server deployment and a WebLogic Server boot failure. For example, attempting to target a JMS server that is using the default file store to a migratable target, will generate the following message:

```
Since the JMS server is targeted to a migratable target, it cannot use the default store.
```

Similar messages are generated for a SAF agent or path service that is targeted to a migratable target and attempts to use the default store. In addition, if the custom store is not targeted to the same migratable target as the migratable service, then the following validation log message will be generated, followed by failed JMS server deployment and a WebLogic Server start failure.

```
The JMS server is not targeted to the same target as its persistent store.
```

Special Considerations When Targeting SAF Agents or Path Service

There are some special targeting choices to consider when targeting SAF agents and a path service to migratable targets. See Targeting Rules for SAF Agents and Targeting Rules for Path Service.

Step 7: Restart the Administration Server and Managed Servers With Modified Migration Policies

You must restart the Administration Server after configuring your JMS services for automatic service migration. Also, you must restart any Managed Servers whose migration policies were modified.

Step 8: Manually Migrate JMS Services Back to the Original Server

You may want to migrate a JMS service back to the original primary server instance once it is back online. Unlike the JTA Transaction Recovery Service, JMS services do not automatically migrate back to the primary server instance when it becomes available, so you need to manually migrate these services.

For instructions on manually migrating the JMS-related services using WLST, see WLST Command and Variable Reference in *WLST Command Reference for WebLogic Server*.

Step 9: Test JMS Migration

To Test if the Automatic JMS Migration is Working

1. Configure:
   - A cluster with two servers Server1 and Server2.
   - Migratable targets MT1 and MT2 with an "Exactly Once" migration policy.
   - File stores on each migratable target named FStore1 and FStore2.
   - JMS servers on each migratable target named JMS1 and JMS2.
   - A JMS system module targeted to the cluster.
   - A subdeployment named SUBD in the system module that references JMS1 and JMS2.
   - A Uniform Distributed Queue in the system module with JNDI name "UDQ" that targets subdeployment SUBD (instead of using default targeting).
2. Boot Server1:
   - Shutdown cluster.
   - Start only Server1.
   - Check Server1's JNDI tree to verify there is a UDQ member with JNDI name "JMS1@UDQ" and another with "JMS2@UDQ".
3. Boot Server2:
   - Start Server2.
   - Note that “JMS1@UDQ” and “JMS2@UDQ” remain on Server1.
   - Manually migrate MT2 back to the Server2.

- Verify that "JMS1@UDQ" and "JMS2@UDQ" are now on Server1 and Server2 respectively (this may take a few seconds).

> **Note**
>
> The cluster targeted HA alternative would automatically migrate JMS2 back to Server2 without a need for manual intervention. For more information, see Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*.

Best Practices for Configuring JMS Migration

Learn about some of the best practices you can use to configure the migratable target during JMS migration.

- In most cases, it is sufficient to use the default migratable target for a server instance. There is one default migratable target per server instance. An alternative is to configure one migratable target per server instance. See Step 3: Configure Migratable Targets.
- Configure one custom store per migratable target and target the store to the migratable target. See Step 4: Configure and Target Custom Stores.
- When configuring JMS services (JMS servers and SAF agents) for each migratable target, ensure that the services refer to the corresponding custom store. Then target the services to each migratable target. See Step 5: Target the JMS Services.
- Ensure that all file stores are configured with a directory that references a shared storage location. This ensures that a store that migrates from one server to another can still load its file data. For similar reasons, migratable database stores should be setup to reference the same data source.
- As you add or remove JMS servers or SAF agents, remember to also add or remove JMS servers or SAF agents to your module subdeployment(s).

For more information, see Best Practices for JMS Beginners and Advanced Users in *Administering JMS Resources for Oracle WebLogic Server*.

Roadmap for Configuring Manual Migration of JMS-Related Services

WebLogic JMS leverages the migration framework by allowing an administrator to specify a migratable target for JMS-related services. Once properly configured, a JMS service can be manually migrated to another WebLogic Server within a cluster using a script without setting up automatic migration. This includes both scheduled migrations as well as manual migrations in response to a WebLogic Server failure within the cluster.

To setup JMS-related services for manual migration on a migratable target within a configured cluster, follow the Roadmap for Configuring Automatic Migration of JMS-Related Services but account for the following differences:

- There is no need to configure a Node Manager.
- There is no need to configure a cluster migration basis.
- Leave the Migration Policy attribute on each Migratable Target at the default of Manual.

Roadmap for Configuring Automatic Migration of the JTA Transaction Recovery Service

The JTA Transaction Recovery Service is designed to gracefully handle transaction recovery after a crash. You can specify to have the Transaction Recovery Service automatically migrated from an unhealthy server instance to a healthy server instance, with the help of the server health monitoring services. This way, the backup server instance can complete transaction work for the failed server instance.

To configure automatic migration of the Transaction Recovery Service for a migratable target within a cluster, perform the following tasks:

Step 1: Configure Managed Servers and Node Manager

Configure the Managed Servers in the cluster for migration, including assigning Managed Servers to a machine. Node Manager must also be running and configured to allow automatic server migration. Node Manager is required for health status information about the server instances involved.

For step-by-step instructions for using the WebLogic Remote Console to complete these tasks, see the following topics in the *Oracle WebLogic Remote Console Online Help*:

- Create Managed Servers

  > **Note**
  >
  > – You must set a unique **Listen Address** value for the Managed Server instance that will host the JTA Transaction Recovery service. Otherwise, the migration will fail.
  >
  > – For information on configuring a primary server instance to not start in Managed Server Independence (MSI) mode, which will prevent concurrent access to the transaction log with another backup server instance in recovery mode, see Managed Server Independence in *Developing JTA Applications for Oracle WebLogic Server*.

- Create and Configure Machines

  > **Note**
  >
  > In general, Machines are required to enable Node Managers to start and stop WebLogic Server instances. When not using a Node Manager, Machines generally are not needed.

- Configure Node Manger, see Step 7. in Configure Machines

> **Note**
>
> For automatic service migration, Consensus leasing requires that you use Node Manager to control server instances within the cluster and that all migratable servers must have a Node Manager instance associated with them. For Database leasing, Node Manager is required only if pre-migration and post-migration scripts are defined. If pre-migration and post-migration scripts are not defined, then Node Manager is not required.

For general information on configuring Node Manager, see Node Manager Overview in *Administering Node Manager for Oracle WebLogic Server*.

Step 2: Configure the Migration Basis

In the Remote Console, on the **Environment**: **Clusters**: *myCluster*: **Migration** page, configure the cluster **Migration Basis** according to how your data persistence environment is configured, selecting either **database** or **consensus**. See Leasing for Migratable Services.

Database Leasing requires that the **Data Source For Automatic Migration** field is set with a reference to valid JDBC system resource. It also requires that a leasing table be created on that resource. See High Availability Database Leasing.

> **Note**
>
> As a best practice, Oracle recommends configuring database leasing instead of consensus leasing.

Step 3: Enable Automatic JTA Migration

In the WebLogic Remote Console, on the **Migratable Targets**: *myManagedServer* **(migratable)**: **General** page, configure the following options:

Select the Automatic JTA Migration Policy

Select the migration policy for services hosted by the JTA migratable target. Valid options are:

- **manual**
- **failure-recovery**
- **shutdown-recovery**

> **Note**
>
> The **exactly-once** migration policy does not apply for JTA.

Configure the automatic migration of the JTA Transaction Recovery Service by selecting a migration policy for services hosted by the JTA migratable target on the **JTA Migratable Target**: **Migration** page, from the **JTA Migration Policy** drop-down menu.

Optionally Select Candidate Servers

You may also want to restrict the potential server instances to which you can migrate the Transaction Recovery Service to those that have access to the current server instance's transaction log files (stored in the default WebLogic store). If no candidate server instances are chosen, then any server instance within the cluster can be chosen as a candidate server instance.

From the Candidate Servers `Available` box, select the Managed Servers that can access the JTA log files. The Managed Servers become valid Candidate Servers when you move them into the `Chosen` box.

> **Note**
>
> You must include the original server instance in the list of chosen server instances so that you can manually migrate the Transaction Recovery Service back to the original server instance, if required.

Optionally Specify Pre/Post-Migration Scripts

You can specify whether you are providing any pre-migration and post-migration scripts to perform any unmounting and mounting of the shared storage, as needed.

- **Pre-Migration Script Path**: It is the path to the pre-migration script to run before a migratable target is actually activated.
- **Post-Migration Script Path**: It is the path to the post-migration script to run after a migratable target is fully deactivated.
- **Post-Migration Script Failure Cancels Automatic Migration**: It specifies whether or not a failure during execution of the post-deactivation script is fatal to the migration.
- **Allow Post-Migration Script To Run On a Different Machine**: It specifies whether or not the post-deactivation script is allowed to run on a different machine.

The pre-migration and post-migration scripts must be located in the *`ORACLE_HOME`*`/user_projects/domains/`*`mydomain`*`/bin/service_migration` directory, where *`mydomain`* is a domain-specific directory, with the same name as the domain. Basic directions for creating pre-migration and post-migration scripts are provided in a `readme.txt` file in this directory.

Step 4: Configure the Default Persistent Store For Transaction Recovery Service Migration

As per Default File Store Availability for JTA, the Transaction Manager uses the default persistent store to store transaction log files. To enable migration of the Transaction Recovery Service, you must configure the default persistent store so that it stores its data files on a persistent storage solution that is available to other server instances in the cluster if the original server instance fails.

Step 5: Restart the Administration Server and Managed Servers With Modified Migration Policies

You must restart the Administration Server after configuring the JTA Transaction Recovery service for automatic service migration.

Also, you must restart any Managed Servers whose migration policies were modified.

Step 6: Automatic Failback of the Transaction Recovery Service Back to the Original Server

After completing transaction recovery for a failed server instance, a backup server instance releases ownership of the Transaction Recovery Service so that the original server instance can reclaim it when the server instance is restarted. If the backup server stops (crashes) for any reason before it completes transaction recovery, its lease will expire. This way when the primary server instance starts, it can reclaim the ownership successfully.

There are two scenarios for automatic failback of the Transaction Recovery Service to the primary server instance:

- Automatic failback *after* recovery is complete:
  - If the backup server instance finishes recovering the log transactions before the primary server instance is restarted, it will initiate an implicit migration of the Transaction Recovery Service back to the primary server instance.
  - For both manual and automatic migration, the post-deactivation script will be executed automatically.
- Automatic failback *before* recovery is complete:
  - If the backup server instance is still recovering the log transactions when the primary server instance is started, during the Transaction Recovery Service initialization of the primary server startup, it will initiate an implicit migration of the Transaction Recovery Service from the backup server instance.

Manual Migration of the JTA Transaction Recovery Service

The JTA Transaction Recovery Service is designed to gracefully handle transaction recovery after a crash. You can manually migrate the Transaction Recovery Service from an unhealthy server instance to a healthy server instance with the help of the server health monitoring services. In this manner, the backup server instance can complete transaction work for the failed server instance.

You can manually migrate the Transaction Recovery Service back to the original server instance by selecting the original server instance as the destination server instance. The backup server instance must not be running when you manually migrate the service back to the original server instance.

Note the following:

- If a backup server instance fails before completing the transaction recovery actions, the primary server instance cannot reclaim ownership of the Transaction Recovery Service and recovery will not be re-attempted on the restarting server instance. Therefore, you must attempt to manually re-migrate the Transaction Recovery Service to another backup server instance.

- If you restart the original server instance while the backup server instance is recovering transactions, the backup server instance will gracefully release ownership of the Transaction Recovery Service. You do not need to stop the backup server instance. For more information, see Recovering Transactions For a Failed Clustered Server in *Developing JTA Applications for Oracle WebLogic Server*.
- For information on configuring a primary backup server instance to not start in Managed Server Independence (MSI) mode, which will prevent concurrent access to the transaction log with another backup server in recovery mode, see Managed Server Independence in *Developing JTA Applications for Oracle WebLogic Server*.

Automatic Migration of User-Defined Singleton Services

WebLogic Server supports the automatic migration of user-defined singleton services.

Automatic singleton service migration allows the automatic health monitoring and migration of singleton services. A singleton service is a service operating within a cluster that is available on only one server instance at any given time. When a migratable service fails or become unavailable for any reason (for example, because of a bug in the service code, server failure, or network failure), it is deactivated at its current location and activated on a new server instance. The process of migrating these services to another server instance is handled using the singleton master. See Singleton Master.

> **Note**
>
> Although the JTA Transaction Recovery Service is also a singleton service that is available on only one node of a cluster at any time, it is configured differently for automatic migration than user-defined singleton services. JMS and JTA services can also be manually migrated. See Understanding the Service Migration Framework.

Overview of Singleton Service Migration

This section provides an overview of how automatic singleton service is implemented in WebLogic Server.

Singleton Master

The singleton master is a lightweight singleton service that monitors other services that can be migrated automatically. The server instance that currently hosts the singleton master is responsible for starting and stopping the migration tasks associated with each migratable service.

> **Note**
>
> Migratable services do not have to be hosted on the same server instance as the singleton master, but they must be hosted within the same cluster.

The singleton master functions similar to the cluster master in that it is maintained by lease competition and runs on only one server instance at a time. Each server instance in a cluster continuously attempts to register the singleton master lease. If the server instance currently

hosting the singleton master fails, the next server instance in the queue will take over the lease and begin hosting the singleton master.

See Cluster Master Role in Whole Server Migration.

> **Note**
>
> The singleton master and cluster master function independently and are not required to be hosted on the same server instance.

The server instance hosting the singleton master maintains a record of all migrations performed, including the target name, source server, destination server, and the timestamp.

Migration Failure

If the migration of a singleton service fails on every candidate server instance within the cluster, the service is left deactivated. You can configure the number of times the singleton master will iterate through the server instances in the cluster.

> **Note**
>
> If you do not explicitly specify a list of candidate server instances, the singleton master will consider all of the cluster members as possible candidates for migration.

Implementing the Singleton Service Interface

A singleton service can be defined either as part of an application or as a standalone service. It is active only on one server instance at any time and so it can be used to perform tasks that you want to be executed on only one member of a cluster.

To create a singleton service, you must create a class that, in addition to any tasks you want the singleton service to perform, it should implement the `weblogic.cluster.singleton.SingletonService` interface.

The `SingletonService` interface contains the following methods, which are used in the process of migration.

- `public void activate()`

  This method should obtain any system resources and start any services required for the singleton service to begin processing requests. This method is called in the following cases:

  - When a newly deployed application is started.
  - During server start.
  - During the activation stage of service migration.

- `public void deactivate()`

  This method is called during server shutdown and the deactivation stage of singleton service migration. This method should release any resources obtained through the `activate()` method. Additionally, it should stop any services that should only be available from one member of a cluster.

> **Note**
>
> MDBs have a lifecycle that is managed with the JMS infrastructure in the domain. Therefore, you cannot use an MDB when implementing the `weblogic.cluster.singleton.SingletonService` interface. For an alternative way to process specific messages one-at-a-time/sequentially within a cluster, see Using the Message Unit-of-Order in *Developing JMS Applications for Oracle WebLogic Server*.

Deploying a Singleton Service and Configuring the Migration Behavior

Depending on how you used the `SingletonService` interface to define a singleton service, you must perform the following steps to deploy it:

- Package and deploy the singleton service within an application (application-scoped).

  Or

- Deploy the singleton service as a standalone service within WebLogic Server (domain-wide).
- Optionally, configure the migration behavior of the singleton service.

The following sections outline these procedures in detail.

Packaging and Deploying a Singleton Service Within an Application

Singleton services that are packaged within an application should have their classes implement the `SingletonService` interface and placed within a JAR file, in `APP-INF/lib` or `APP-INF/classes`, or within an EAR-level `lib` directory. JNDI binding for application-scoped singleton services is done programmatically in the `SingletonService` interface. The life cycle of an application-scoped singleton service is tied with the life cycle of the application.

For standalone singleton services, their classes should be made available in the WebLogic Server system classpath.

Also, add the following entry to the `weblogic-application.xml` descriptor file:

```
<weblogic-application>
...
   <singleton-service>
      <class-name>mypackage.MySingletonServiceImpl</class-name>
      <name>Appscoped_Singleton_Service</name>
   </singleton-service>
...
</weblogic-application>
```

> **Note**
>
> The `<class-name>` and `<name>` elements are required.

Deployment of an application-scoped singleton service occurs automatically as part of the application deployment. The candidate server instances for the singleton service will be the cluster members where the application is deployed.

Deploying a Singleton Service as a Standalone Service in WebLogic Server

After you have created a singleton service class using the `SingletonService` interface, you must define it as a singleton service within WebLogic Server. This singleton service object contains the following information:

- The path to the class to load as the singleton service.
- The preferred server instance and other candidate server instances for the singleton service.

The following excerpt from the `<cluster>` element of `config.xml` file shows how a singleton service is defined:

```
<singleton-service>
   <name>SingletonTestServiceName</name>
   <user-preferred-server>myManaged1</user-preferred-server>
   <class-name>mycompany.myprogram.subpackage.SingletonTestServiceImpl</class-name>
   <cluster>myCluster</cluster>
</singleton-service>
```

Configuring Singleton Service Migration

A singleton service is automatically configured to be an `exactly-once` service, which indicates that if at least one Managed Server in the candidate list is running, then the service will be active somewhere in the cluster. You can modify certain singleton service migration parameters using the following methods:

- WebLogic Remote Console: It allows you to create and configure singleton services. In the **Edit Tree**, go to **Environment**, then **Singleton Services**.
- WebLogic Scripting Tool (WLST): It allows you to configure automatic service migration using the `MigratableTargetManagementMBean`. See WLST Command and Variable Reference in *WLST Command Reference for WebLogic Server*.

8

Cluster Architectures

Learn about the alternative architectures for an Oracle WebLogic Server cluster.

This chapter includes the following sections:

Architectural and Cluster Terminology

Learn the terms to describe clusters and their architecture.

The following sections define the terms used in this document.

Architecture

In this context, the *architecture* refers to how the application tiers are deployed to one or more clusters.

Web Application Tiers

A Web application is divided into several "tiers" that correspond to the logical services the application provides. Because not all Web applications are alike, your application may not utilize all of the tiers described below. Also keep in mind that the tiers represent logical divisions of an application's services, and not necessarily physical divisions between hardware or software components. In some cases, a single machine running a single WebLogic Server instance can provide all of the tiers described below.

- Web Tier

  The *Web tier* provides static content (for example, simple HTML pages) to clients of a Web application. The Web tier is generally the first point of contact between external clients and the Web application. A simple Web application may have a Web tier that consists of one or more machines running Apache or Microsoft Internet Information Server.

- Presentation Tier

  The *presentation tier* provides dynamic content (for example, servlets or Jakarta Server Pages) to clients of a Web application. A cluster of WebLogic Server instances that hosts servlets and/or JSPs comprises the presentation tier of a Web application. If the cluster also serves static HTML pages for your application, it encompasses both the Web tier and the presentation tier.

- Object Tier

  The *object tier* provides Java objects (for example, Jakarta Enterprise Beans or RMI classes) and their associated business logic to a Web application. A WebLogic Server cluster that hosts EJBs provides an object tier.

Combined Tier Architecture

A cluster architecture in which all tiers of the Web application are deployed to a single WebLogic Server cluster is called a combined tier architecture.

De-Militarized Zone (DMZ)

The *De-Militarized Zone (DMZ)* is a logical collection of hardware and services that is made available to outside, untrusted sources. In most Web applications, a bank of Web servers resides in the DMZ to allow browser-based clients access to static HTML content.

The DMZ may provide security against outside attacks to hardware and software. However, because the DMZ is available to untrusted sources, it is less secure than an internal system. For example, internal systems may be protected by a firewall that denies all outside access. The DMZ may be protected by a firewall that *hides* access to individual machines, applications, or port numbers, but it still permits access to those services from untrusted clients.

Load Balancer

In this document, the term *load balancer* describes any technology that distributes client connection requests to one or more distinct IP addresses. For example, a simple Web application may use the DNS round-robin algorithm as a load balancer. Larger applications generally use hardware-based load balancing solutions such as those from Alteon WebSystems, which may also provide firewall-like security capabilities.

Load balancers provide the capability to associate a client connection with a particular server in the cluster, which is required when using in-memory replication for client session information. With certain load balancing products, you must configure the cookie persistence mechanism to avoid overwriting the WebLogic Server cookie which tracks primary and secondary servers used for in-memory replication. For more information, see Load Balancing HTTP Sessions with an External Load Balancer.

Proxy Plug-In

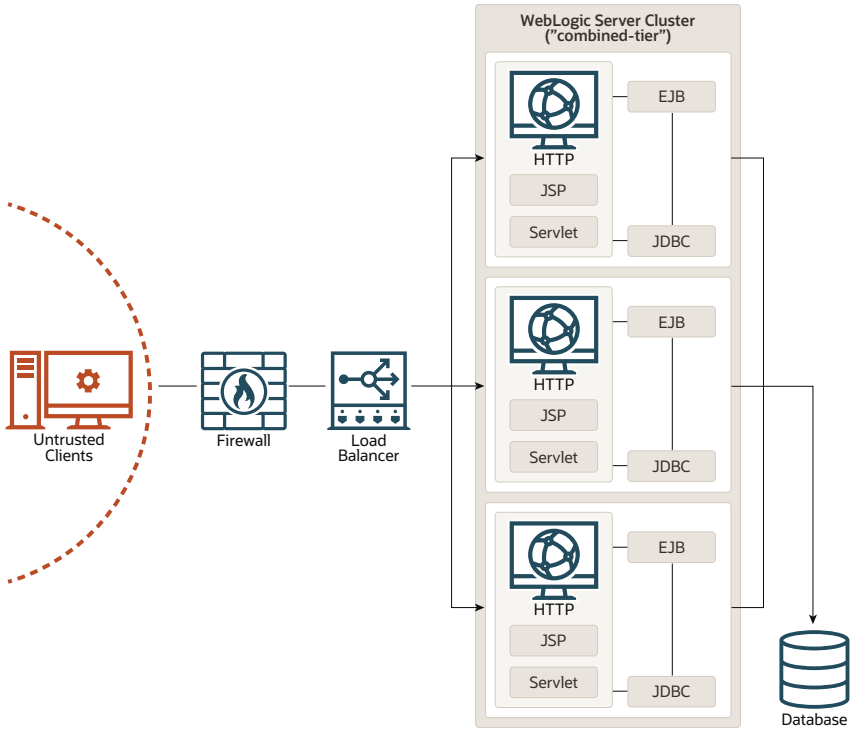
A *proxy plug-in* is a WebLogic Server extension to an HTTP server such as Apache or Microsoft Internet Information Server that accesses clustered servlets provided by a WebLogic Server cluster. The proxy plug-in contains the load balancing logic for accessing servlets and JSPs in a WebLogic Server cluster. Proxy plug-ins also contain the logic for accessing the replica of a client's session state if the primary WebLogic Server hosting the session state fails.

Recommended Basic Architecture

The recommended basic architecture is a combined tier architecture—all tiers of the Web application are deployed to the same WebLogic Server cluster.

This architecture is illustrated in Figure 8-1, below.

Figure 8-1 Recommended Basic Architecture



The benefits of the recommended basic architecture are:

- Ease of administration

  Because a single cluster hosts static HTTP pages, servlets, and EJBs, you can configure the entire Web application and deploy/undeploy objects using the WebLogic Remote Console. You do not need to maintain a separate bank of Web servers (and configure WebLogic Server proxy plug-ins) to benefit from clustered servlets.

- Flexible load balancing

  Using load balancing hardware directly in front of the WebLogic Server cluster enables you to use advanced load balancing policies for accessing both HTML and servlet content. For example, you can configure your load balancer to detect current server loads and direct client requests appropriately.

- Robust security

  Placing a firewall in front of your load balancing hardware enables you to set up a DMZ for your Web application using minimal firewall policies.

- Optimal performance

  The combined tier architecture offers the best performance for applications in which most or all of the servlets or JSPs in the presentation tier typically access objects in the object tier, such as EJBs.

> **Note**
>
> When using a third-party load balancer with in-memory session replication, you must ensure that the load balancer maintains a client's connection to the WebLogic Server instance that hosts its primary session state (the point-of-contact server). For more information, see Load Balancing HTTP Sessions with an External Load Balancer.

When Not to Use a Combined Tier Architecture

While a combined tier architecture, such as the Recommended Basic Architecture, meets the needs of many Web applications, it limits your ability to fully employ the load balancing and failover capabilities of a cluster. Load balancing and failover can be introduced only at the interfaces between Web application tiers. So, when tiers are deployed to a single cluster, you can only load balance between clients and the cluster.

Because most load balancing and failover occurs between clients and the cluster itself, a combined tier architecture meets the needs of most Web applications.

However, combined-tier clusters provide no opportunity for load balancing method calls to clustered EJBs. Because clustered objects are deployed on all WebLogic Server instances in the cluster, each object instance is available locally to each server. WebLogic Server optimizes method calls to clustered EJBs by always selecting the local object instance, rather than distributing requests to remote objects and incurring additional network overhead.

In most cases, this collocation strategy is more efficient than load balancing each method request to a different server. However, if the processing load to individual servers becomes unbalanced, it may eventually become more efficient to submit method calls to remote objects rather than process methods locally.

To utilize load balancing for method calls to clustered EJBs, you must split the presentation and object tiers of the Web application onto separate physical clusters, as described in the following section.

Consider the frequency of invocations of the object tier by the presentation tier when deciding between a combined tier and multitier architecture. If presentation objects usually invoke the object tier, a combined tier architecture may offer better performance than a multitier architecture.

Recommended Multitier Architecture

Learn the recommended multitier architecture, in which different tiers of your application are deployed to different clusters.

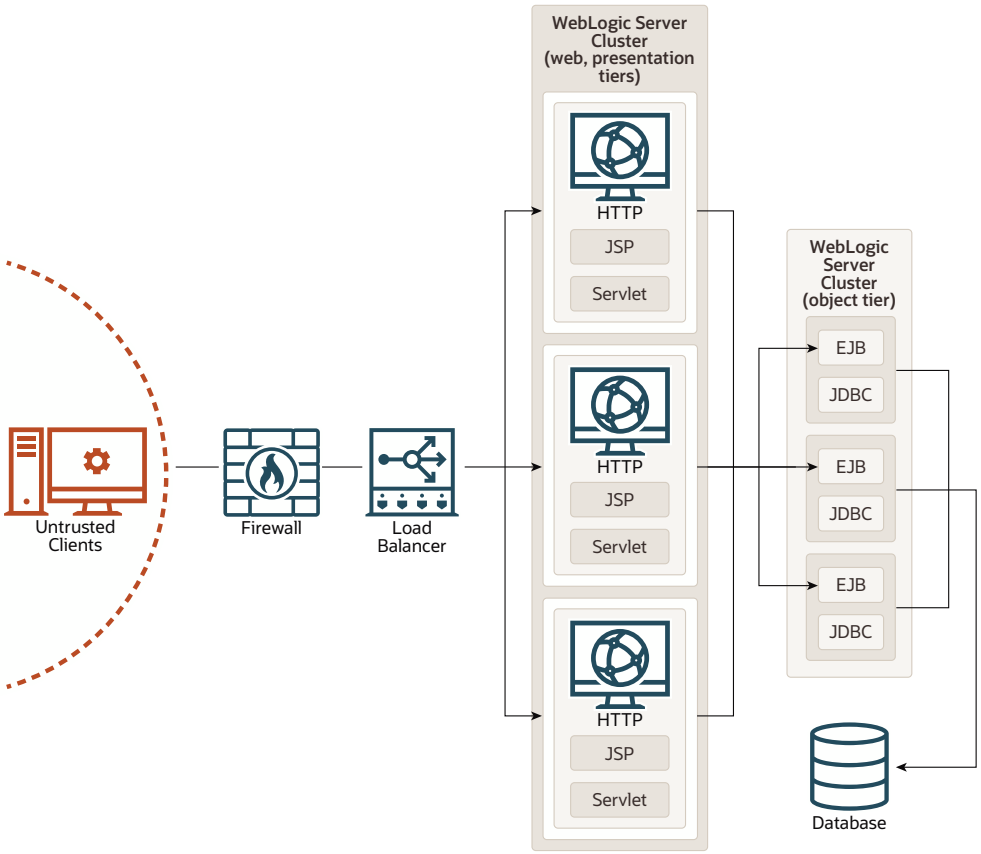
The recommended multitier architecture uses two separate WebLogic Server clusters: one to serve static HTTP content and clustered servlets, and one to serve clustered EJBs. The multitier cluster is recommended for Web applications that:

- Require load balancing for method calls to clustered EJBs.
- Require more flexibility for balancing the load between servers that provide HTTP content and servers that provide clustered objects.
- Require higher availability (fewer single points of failure).

> **Note**
>
> Consider the frequency of invocations from the presentation tier to the object tier when considering a multitier architecture. If presentation objects usually invoke the object tier, a combined tier architecture may offer better performance than a multitier architecture.

Figure 8-2 depicts the recommended multitier architecture.

Figure 8-2 Recommended Multitier Architecture



Physical Hardware and Software Layers

In the recommended multitier architecture, the application tiers are hosted on two separate physical layers of hardware and software.

Web/Presentation Layer

The Web/presentation layer consists of a cluster of WebLogic Server instances dedicated to hosting static HTTP pages, servlets, and JSPs. This servlet cluster *does not* host clustered objects. Instead, servlets in the presentation tier cluster act as clients for clustered objects, which reside on an separate WebLogic Server cluster in the object layer.

Object Layer

The object layer consists of a cluster of WebLogic Server instances that hosts only clustered objects such as EJBs and RMI objects as necessary for the Web application. By hosting the object tier on a dedicated cluster, you lose the default collocation optimization for accessing clustered objects described in Optimization for Collocated Objects. However, you gain the ability to load balance on each method call to certain clustered objects, as described in the following section.

Benefits of Multitier Architecture

The multitier architecture provides these advantages:

- Load Balancing EJB Methods

  By hosting servlets and EJBs on separate clusters, servlet method calls to EJBs can be load balanced across multiple servers. For more information, see Load Balancing Clustered Objects in a in Multitier Architecture.

- Improved Server Load Balancing

  Separating the presentation and object tiers onto separate clusters provides more options for distributing the load of the Web application. For example, if the application accesses HTTP and servlet content more often than EJB content, you can use a large number of WebLogic Server instances in the presentation tier cluster to concentrate access to a smaller number of servers hosting EJBs.

- Higher Availability

  By utilizing additional WebLogic Server instances, the multitier architecture has fewer points of failure than the basic cluster architecture. For example, if a WebLogic Server that hosts EJBs fails, the HTTP and servlet-hosting capacity of the Web application is not affected.

- Improved Security Options

  By separating the presentation and object tiers onto separate clusters, you can use a firewall policy that places only the servlet/JSP cluster in the DMZ. Servers hosting clustered objects can be further protected by denying direct access from untrusted clients. For more information, see Security Options for Cluster Architectures.

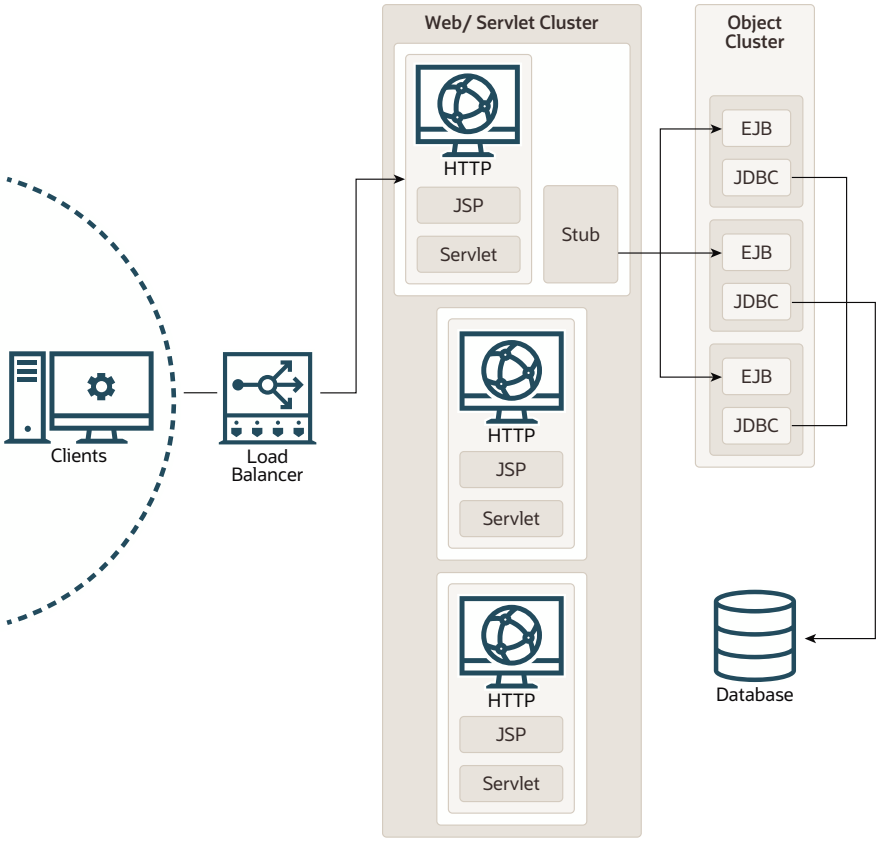
Load Balancing Clustered Objects in a in Multitier Architecture

WebLogic Server's collocation optimization for clustered objects relies on having a clustered object (the EJB or RMI class) hosted on the same server instance as the replica-aware stub that calls the object.

The net effect of isolating the object tier is that no client (HTTP client, Java client, or servlet) ever acquires a replica-aware stub on the same server that hosts the clustered object. Because of this, WebLogic Server cannot use its collocation optimization and servlet calls to clustered objects are automatically load balanced according to the logic contained in the replica-aware stub. For more information, see Optimization for Collocated Objects.

Figure 8-3 depicts a client accessing a clustered EJB instance in the multitier architecture.

Figure 8-3 Load Balancing Objects in a Multitier Architecture



By tracing the path of the client connection, you can see the implication of isolating the object tier onto separate hardware and software.

1. An HTTP client connects to one of several WebLogic Server instances in the Web/servlet cluster, going through a load balancer to reach the initial server.
2. The client accesses a servlet hosted on the WebLogic Server cluster.
3. The servlet acts as a client to clustered objects required by the Web application. In the example above, the servlet accesses a stateless session EJB.

   The servlet looks up the EJB on the WebLogic Server cluster that hosts clustered objects. The servlet obtains a replica-aware stub for the bean, which lists the addresses of all servers that host the bean, as well as the load balancing logic for accessing bean replicas.

> **Note**
>
> EJB replica-aware stubs and EJB home load algorithms are specified using elements of the EJB deployment descriptor. See weblogic-ejb-jar.xml Deployment Descriptor Reference in *Developing Enterprise JavaBeans, Version 2.1, for Oracle WebLogic Server for more information*.

4. When the servlet next accesses the EJB (for example, in response to another client), it uses the load-balancing logic present in the bean's stub to locate a replica. In the example above, multiple method calls are directed using the round-robin algorithm for load balancing.

In this example, if the same WebLogic Server cluster hosted both servlets and EJBs (as in Recommended Basic Architecture), WebLogic Server would not load balance requests for the EJB. Instead, the servlet would always invoke methods on the EJB replica hosted on the local server. Using the local EJB instance is more efficient than making remote method calls to an EJB on another server. However, the multitier architecture enables remote EJB access for applications that require load balancing for EJB method calls.

Configuration Considerations for Multitier Architecture

A multitier architecture may require adjustments to the configuration, as described in the following sections.

IP Socket Usage

Because the multitier architecture provides load balancing for clustered object calls, the system generally utilizes more IP sockets than a combined-tier architecture. In particular, during peak socket usage, each WebLogic Server instance in the cluster that hosts servlets and JSPs may potentially use a maximum of:

- One socket for replicating HTTP session states between primary and secondary servers.
- In addition, one socket for each WebLogic Server in the EJB cluster, for accessing remote objects.

For example, in Figure 8-2, each server in the servlet/JSP cluster could potentially open a maximum of five sockets. This maximum represents a worst-case scenario where primary and secondary session states are equally dispersed throughout the servlet cluster, and each server in the servlet cluster simultaneously accesses a remote object on each server in the object cluster. In most cases, the number of actual sockets in use would be less than this maximum.

If you use a pure-Java sockets implementation with the multitier architecture, ensure that you configure enough socket reader threads to accommodate the maximum potential socket usage. For more information, see Configuring Reader Threads for Java Socket Implementation in *Tuning Performance of Oracle WebLogic Server*

.

Hardware Load Balancers

Because the multitier architecture uses a hardware load balancer, you must configure the load balancer to maintain a "sticky" connection to the client's point-of-contact server if you use in-memory session state replication. See Configure Load Balancing Method for EJBs and RMIs .

Limitations of Multitier Architectures

This section summarizes the limitations of multitier cluster architectures.

No Collocation Optimization

Because the recommended multitier architecture cannot optimize object calls using the collocation strategy, the Web application incurs network overhead for all method calls to clustered objects. This overhead may be acceptable, however, if your Web application requires any of the benefits described in Benefits of Multitier Architecture.

For example, if your Web clients make heavy use of servlets and JSPs but access a relatively small set of clustered objects, the multitier architecture enables you to concentrate the load of servlets and object appropriately. You may configure a servlet cluster of ten WebLogic Server

instances and an object cluster of three WebLogic Server instances, while fully utilizing each server's processing power.

Firewall Restrictions

If you place a firewall between the servlet cluster and object cluster in a multitier architecture, you must bind all servers in the object cluster to public DNS names, rather than IP addresses. Binding those servers with IP addresses can cause address translation problems and prevent the servlet cluster from accessing individual server instances.

If the internal and external DNS names of a WebLogic Server instance are not identical, use the `ExternalDNSName` attribute for the server instance to define the server's external DNS name. Outside the firewall the `ExternalDNSName` should translate to external IP address of the server.

Use of `ExternalDNSName` is required for configurations in which a firewall is performing Network Address Translation, unless clients are accessing WebLogic Server using t3 and the default channel. For instance, `ExternalDNSName` is required for configurations in which a firewall is performing Network Address Translation, and clients are accessing WebLogic Server using HTTP through a proxy plug-in.

Recommended Proxy Architectures

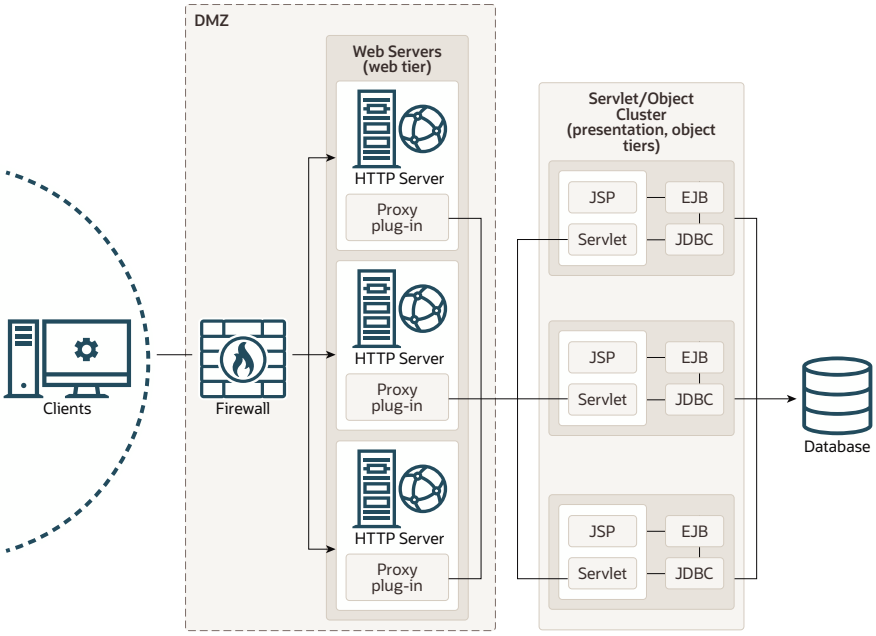
You can configure WebLogic Server clusters to operate alongside existing Web servers. In such an architecture, a bank of Web servers provides static HTTP content for the Web application, using a WebLogic proxy plug-in or `HttpClusterServlet` to direct servlet and JSP requests to a cluster.

The following sections describe two alternative proxy architectures:

Two-Tier Proxy Architecture

The two-tier proxy architecture illustrated in Figure 8-4 is similar to the Recommended Basic Architecture, except that static HTTP servers are hosted on a bank of Web servers.

Figure 8-4 Two-Tier Proxy Architecture



Physical Hardware and Software Layers

The two-tier proxy architecture contains two physical layers of hardware and software.

Web Layer

The proxy architecture utilizes a layer of hardware and software dedicated to the task of providing the application's Web tier. This physical Web layer can consist of one or more identically-configured machines that host one of the following application combinations:

- WebLogic Server with the `HttpClusterServlet`.
- Apache with the Apache HTTP Server (proxy) plug-in
- Microsoft Internet Information Server with the WebLogic Server Microsoft-IIS proxy plug-in.

Regardless of which Web server software you select, remember that the physical tier of Web servers should provide only static Web pages. Dynamic content such as servlets and JSPs are proxied via the proxy plug-in or `HttpClusterServlet` to a WebLogic Server cluster that hosts servlets and JSPs for the presentation tier.

Servlet/Object Layer

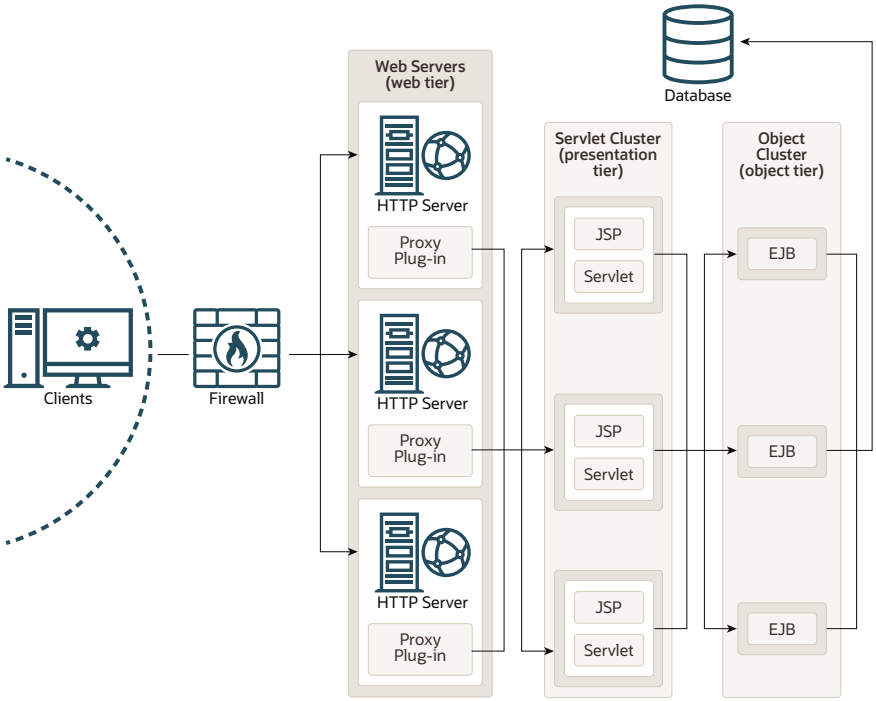
The recommended two-tier proxy architecture hosts the presentation and object tiers on a cluster of WebLogic Server instances. This cluster can be deployed either on a single machine or on multiple separate machines.

The Servlet/Object layer differs from the combined-tier cluster described in Recommended Basic Architecture. In that, it does not provide static HTTP content to application clients.

Multitier Proxy Architecture

You can also use a bank of Web servers as the front-end to a pair of WebLogic Server clusters that host the presentation and object tiers. Figure 8-5 shows this architecture.

Figure 8-5 Multitier Proxy Architecture



This architecture provides the same benefits (and the same limitations) as the Recommended Multitier Architecture. It differs only insofar as the Web tier is placed on a separate bank of Web servers that utilize WebLogic proxy plug-ins.

Proxy Architecture Benefits

Using standalone Web servers and proxy plug-ins provides the following advantages:

- Utilize Existing Hardware

  If you already have a Web application architecture that provides static HTTP content to clients, you can easily integrate existing Web servers with one or more WebLogic Server clusters to provide dynamic HTTP and clustered objects.

- Familiar Firewall Policies

  Using a Web server proxy at the front-end of your Web application enables you to use familiar firewall policies to define your DMZ. In general, you can continue placing the Web servers in your DMZ while disallowing direct connections to the remaining WebLogic Server clusters in the architecture. Figure 8-5 depict this DMZ policy.

Proxy Architecture Limitations

Using standalone Web servers and proxy plug-ins limits your Web application in the following ways:

- Additional administration

  The Web servers in the proxy architecture must be configured using third-party utilities, and do not appear within the WebLogic Server administrative domain. You must also install and configure WebLogic proxy plug-ins to the Web servers in order to benefit from clustered servlet access and failover.

- Limited Load Balancing Options

  When you use proxy plug-ins or the `HttpClusterServlet` to access clustered servlets, the load balancing algorithm is limited to a simple round-robin strategy. For more information, see Round-Robin Load Balancing.

Proxy Plug-In Versus Load Balancer

Using a load balancer directly with a WebLogic Server cluster provides several benefits over proxying servlet requests. First, using WebLogic Server with a load balancer requires no additional administration for client setup. You do not need to set up and maintain a separate layer of HTTP servers, and you do not need to install and configure one or more proxy plug-ins. Removing the Web proxy layer also reduces the number of network connections required to access the cluster.

Using load balancing hardware provides more flexibility for defining load balancing algorithms that suit the capabilities of your system. You can use any load balancing strategy (for example, load-based policies) that your load balancing hardware supports. With proxy plug-ins or the `HttpClusterServlet`, you are limited to a simple round-robin algorithm for clustered servlet requests.

> **Note**
>
> Using a third-party load balancer may require additional configuration if you use in-memory session state replication. In this case, you must ensure that the load balancer maintains a "sticky" connection between the client and its point-of-contact server, so that the client accesses the primary session state information. When using proxy plug-ins, no special configuration is necessary because the proxy automatically maintains a sticky connection.

Security Options for Cluster Architectures

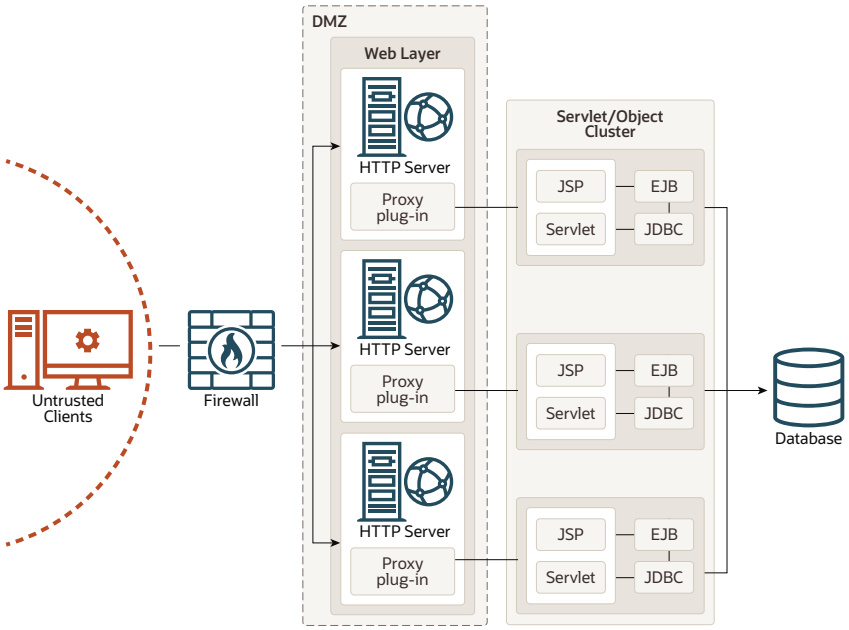
The boundaries between physical hardware and software layers in the recommended configurations provide potential points for defining your Web application's DMZ. However, not all boundaries can support a physical firewall, and certain boundaries can support only a subset of typical firewall policies.

The sections that follow describe several common ways of defining your DMZ to create varying levels of application security.

Basic Firewall for Proxy Architectures

The basic firewall configuration uses a single firewall between untrusted clients and the Web server layer, and it can be used with either the Recommended Basic Architecture or Recommended Multitier Architecture cluster architectures.

Figure 8-6 Basic Proxy with Firewall Architecture



In the configuration shown in Figure 8-6, above, the single firewall can use any combination of policies (application-level restrictions, NAT, IP masquerading) to filter access to three HTTP servers. The most important role for the firewall is to deny direct access to any other servers in the system. In other words, the servlet layer, the object layer, and the database itself must not be accessible from untrusted clients.

> **Note**
>
> You can place the physical firewall either in front of the Web servers or behind it in the DMZ. Placing the firewall in front of the Web servers simplifies your firewall policies, because you need only permit access to the Web servers and deny access to all other systems.

Firewall Between Proxy Layer and Cluster

If you place a firewall between the proxy layer and the cluster, follow these configuration guidelines:

- Bind to clustered server instances using publicly-listed DNS names, rather than IP addresses, to ensure that the proxy plug-ins can connect to each server in the cluster without address translation error that might otherwise occur, as described in Firewall Considerations.

- If the internal and external DNS names of a clustered server instance are not identical, use the `ExternalDNSName` attribute for the server instance to define the external DNS name. Outside the firewall the `ExternalDNSName` should translate to external IP address of the server instance.

> **Note**
>
> If the clustered servers segregate HTTPS and HTTP traffic on a pair of custom channels, see Channels, Proxy Servers, and Firewalls in *Administering Server Environments for Oracle WebLogic Server*.

DMZ with Basic Firewall Configurations

By denying access to all except the Web server layer, the basic firewall configuration creates a small footprint for DMZ that includes only three Web servers. However, a more conservative DMZ definition might take into account the possibility that a malicious client may gain access to servers hosting the presentation and object tiers.

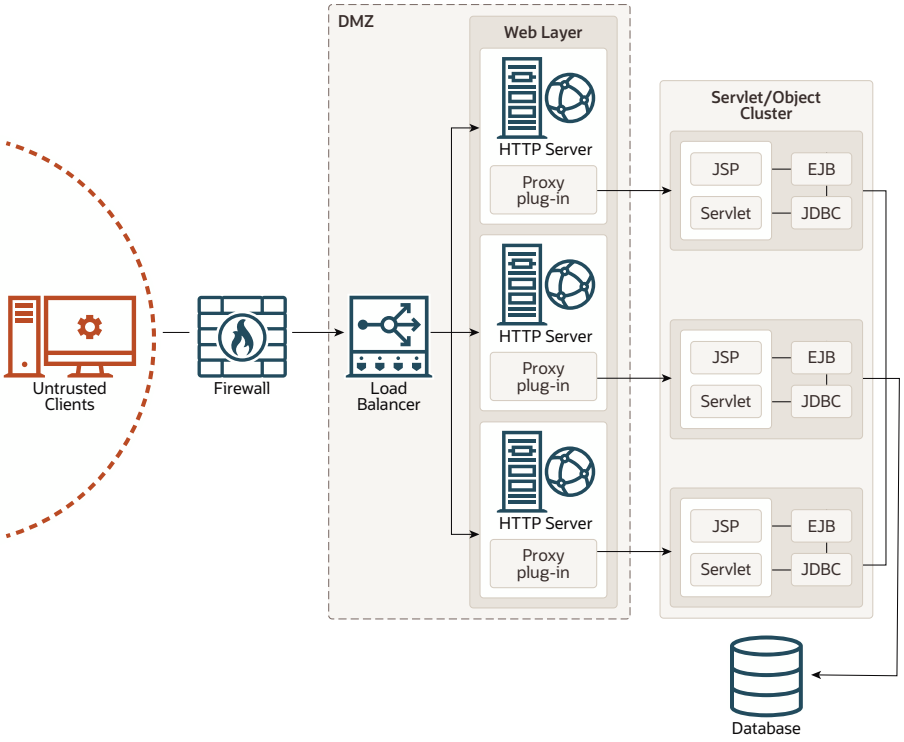
For example, assume that a hacker gains access to one of the machines hosting a Web server. Depending on the level of access, the hacker may then be able to gain information about the proxied servers that the Web server accesses for dynamic content.

If you choose to define your DMZ more conservatively, you can place additional firewalls using the information in Additional Security for Shared Databases.

Combining Firewall with Load Balancer

If you use load balancing hardware with a recommended cluster architecture, you must decide how to deploy the hardware in relationship to the basic firewall. Although many hardware solutions provide security features in addition to load balancing services, most sites rely on a firewall as the first line of defense for their Web applications. In general, firewalls provide the most well-tested and familiar security solution for restricting Web traffic, and should be used in front of load balancing hardware, as shown in Figure 8-7, below.

Figure 8-7 Basic Proxy with Firewall and Load Balancer Architecture



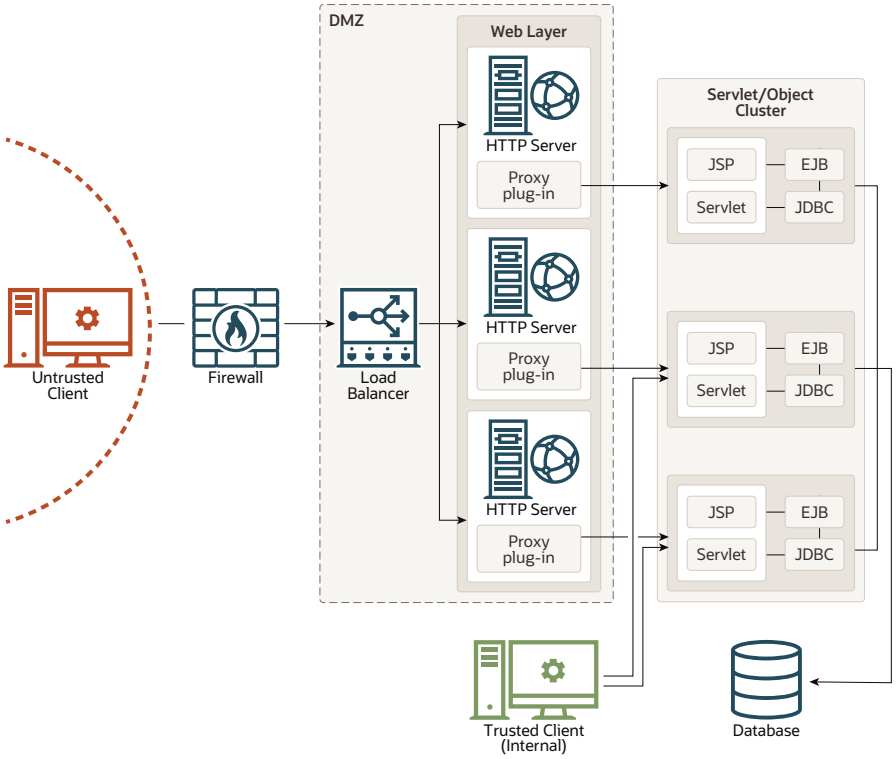
The above setup places the load balancer within the DMZ along with the Web tier. Using a firewall in this configuration can simplify security policy administration because the firewall needs only limited access to the load balancer. This setup can also simplify administration for sites that support internal clients to the Web application, as described below.

Expanding the Firewall for Internal Clients

If you support internal clients that require direct access to your Web application (for example, remote machines that run proprietary Java applications), you can expand the basic firewall configuration to allow restricted access to the presentation tier. The way in which you expand access to the application depends on whether you treat the remote clients as trusted or untrusted connections.

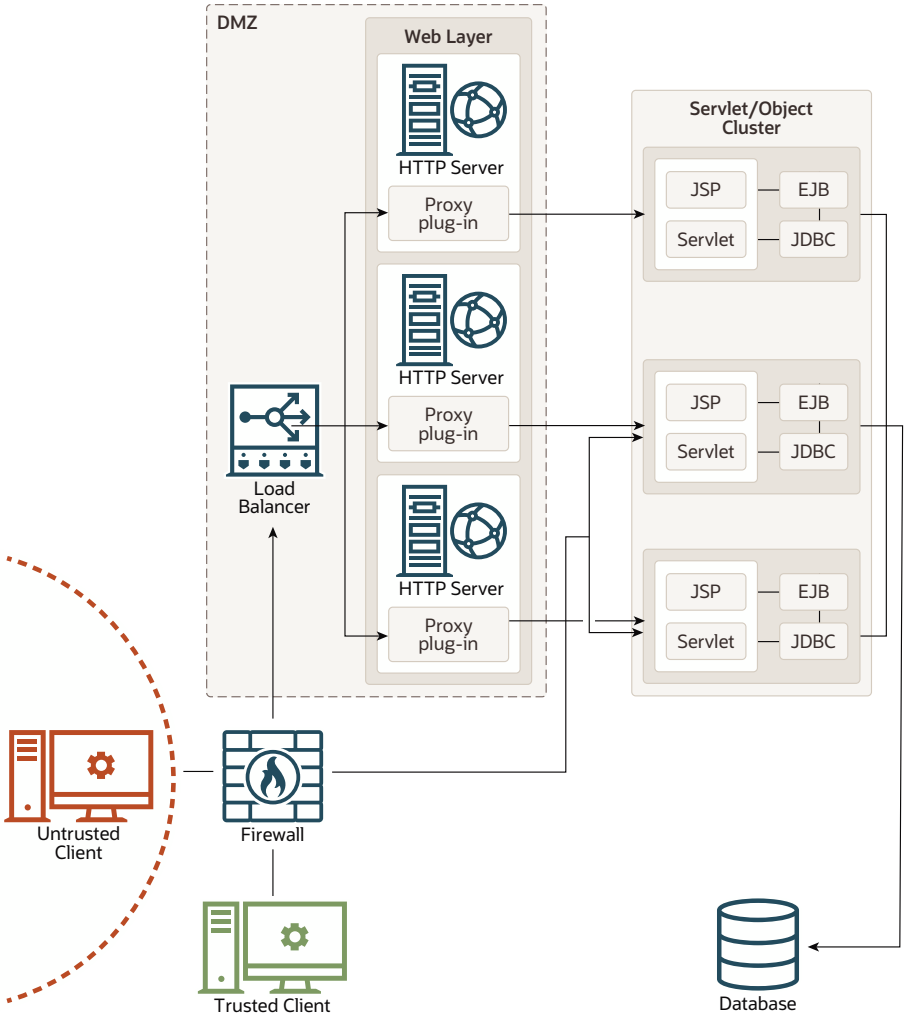
If you use a Virtual Private Network (VPN) to support remote clients, the clients may be treated as trusted connections and can connect directly to the presentation tier going through a firewall. This configuration is shown in Figure 8-8, below.

Figure 8-8 VPN Users have Restricted Access Through Firewall



If you do not use a VPN, all connections to the Web application (even those from remote sites using proprietary client applications) should be treated as untrusted connections. In this case, you can modify the firewall policy to permit application-level connections to WebLogic Server instances hosting the presentation tier, as shown in Figure 8-9.

Figure 8-9 Application Components Have Restricted Access Through Firewall



Additional Security for Shared Databases

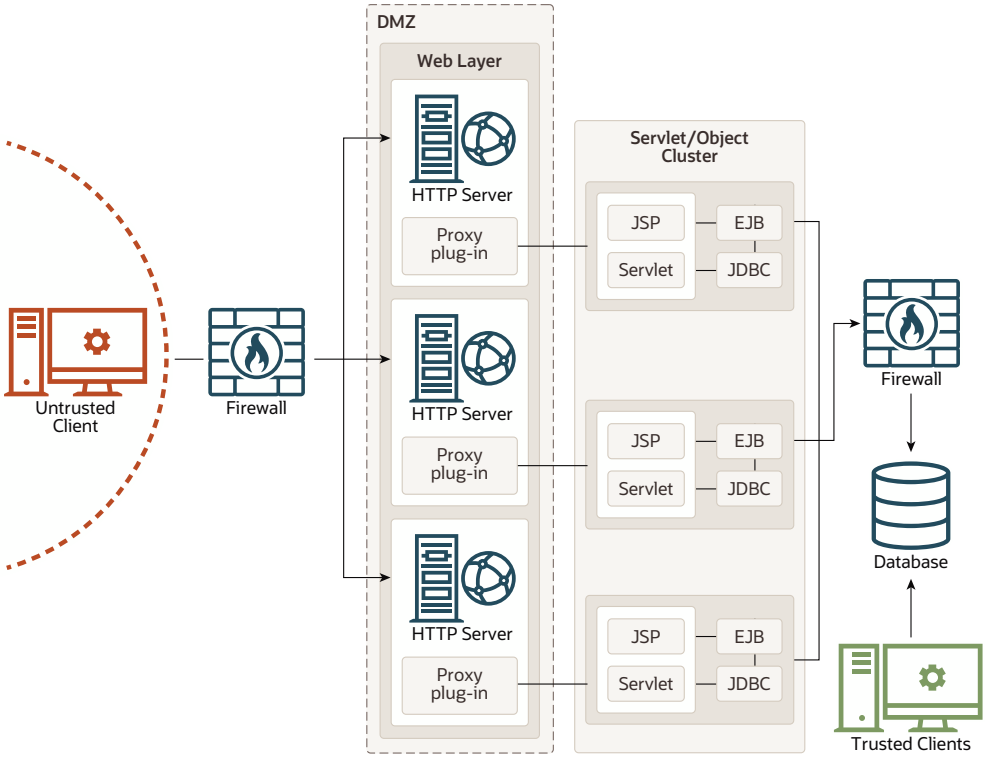
If you use a single database that supports both internal data and data for externally-available Web applications, you should consider placing a hard boundary between the object layer that accesses your database. Doing so simply reinforces the DMZ boundaries described in Basic Firewall for Proxy Architectures, by adding an additional firewall.

DMZ with Two Firewall Configuration

The configuration shown in Figure 8-10 places an additional firewall in front of a database server that is shared by the Web application and internal (trusted) clients. This configuration provides additional security when the first firewall is breached, and a hacker ultimately gains access to servers hosting the object tier.

> **Note**
>
> This circumstance should be extremely unlikely in a production environment. Your site should have the capability to detect and stop a malicious break-in long before a hacker gains access to machines in the object layer.

Figure 8-10 DMZ with Two Firewalls Architecture



In the above configuration, the boundary between the object tier and the database is hardened using an additional firewall. The firewall maintains a strict application-level policy that denies access to all connections except JDBC connections from WebLogic Servers hosting the object tier.

9

Setting Up WebLogic Clusters

Learn the guidelines and instructions for configuring an Oracle WebLogic Server cluster.

This chapter includes the following sections:

Before You Start

Learn the prerequisite tasks for setting up a WebLogic Server cluster.

Understand the Configuration Process

If you have a basic understanding of the cluster configuration process and how to accomplish the configuration tasks, information in this section will be most useful.

For information about the configuration facilities available in WebLogic Server and the tasks they support, see Understanding Cluster Configuration.

Determine Your Cluster Architecture

Determine the best cluster architecture that suits your needs. Key architectural decisions include:

- Should you combine all application tiers in a single cluster or segment your application tiers in separate clusters?
- How will you balance the load among server instances in your cluster? Will you:
  - Use basic WebLogic Server load balancing,
  - Implement a third-party load balancer, or
  - Deploy the Web tier of your application on one or more secondary HTTP servers, and proxy requests to it?
- Should you define your Web applications DMZ with one or more firewalls?

To guide these decisions, see Cluster Architectures, and Load Balancing in a Cluster.

The architecture you choose affects how you set up your cluster. The cluster architecture may also require that you install or configure other resources, such as load balancers, HTTP servers, and proxy plug-ins.

Consider Your Network and Security Topologies

Your security requirements form the basis for designing the appropriate security topology. For a discussion of several alternative architectures that provide varying levels of application security, see Security Options for Cluster Architectures.

> **Note**
>
> Some network topologies can interfere with multicast communication. If you are deploying a cluster across a WAN, see If Your Cluster Spans Multiple Subnets In a WAN .
>
> Avoid deploying server instances in a cluster across a firewall. For a discussion of the impact of tunneling multicast traffic through a firewall, see Firewalls Can Break Multicast Communication.

Choose Machines for the Cluster Installation

Identify the machine or machines where you plan to install WebLogic Server (throughout this section we refer to such machines as "hosts") and ensure that they have the resources required. WebLogic Server allows you to set up a cluster on a single, non-multihomed machine. This new capability is useful for demonstration or development environments.

> **Note**
>
> Do not install WebLogic Server on machines that have dynamically assigned IP addresses.

WebLogic Server Instances on Multi-CPU Machines

WebLogic Server has no built-in limit for the number of server instances that can reside in a cluster. Large, multi-processor servers such as Sun Microsystems, Inc. Sun Enterprise 10000 can host very large clusters or multiple clusters.

Oracle recommends that you start with one server per CPU and then scale up based on the expected behavior. However, as with all capacity planning, you should test the actual deployment with your target Web applications to determine the optimal number and distribution of server instances. For more information see Running Multiple Server Instances on Multi-Core Machines in *Tuning Performance of Oracle WebLogic Server* for additional information.

Check Host Machines' Socket Reader Implementation

For best socket performance, configure the host to use the native socket reader implementation for your operating system, rather than the pure-Java implementation. For understanding and the instructions about configuring native sockets or optimizing pure-Java socket communications, see Peer-to-Peer Communication Using IP Sockets.

Setting Up a Cluster on a Disconnected Windows Machine

If you want to demonstrate a WebLogic Server cluster on a single, disconnected Windows machine, you must force Windows to load the TCP/IP stack. By default, Windows does not load the TCP/IP stack if it does not detect a physical network connection.

To force Windows to load the TCP/IP stack, disable the Windows media sensing feature using the instructions in `How to Disable Media Sense for TCP/IP in Windows`.

Identify Names and Addresses

During the cluster configuration process, you supply addressing information, such as IP addresses or DNS names and port numbers for the server instances in the cluster.

For information on intra-cluster communication, and how it enables load balancing and failover, see Choosing WebLogic Server Cluster Messaging Protocols.

When you set up your cluster, you must provide location information for:

- Administration Server
- Managed Servers
- Multicast location

Read the sections that follow for an explanation of the information you must provide, and factors that influence the method you use to identify resources.

Avoiding Listen Address Problems

As you configure a cluster, you can specify address information using either IP addresses or DNS names.

DNS Names or IP Addresses?

Consider the purpose of the cluster when deciding whether to use DNS names or IP addresses. For production environments, the use of DNS names is generally recommended. The use of IP addresses can result in translation errors, if:

- Clients will connect to the cluster through a firewall, or
- You have a firewall between the presentation and object tiers. For example, you have a servlet cluster and EJB cluster with a firewall in between, as described in the recommended multitier cluster.

You can avoid translation errors by binding the address of an individual server instance to a DNS name. Make sure that a server instance's DNS name is identical on each side of firewalls in your environment, and do not use a DNS name that is also the name of an NT system on your network.

For more information about using DNS names instead of IP addresses, see Firewall Considerations.

When Internal and External DNS Names Vary

If the internal and external DNS names of a WebLogic Server instance are not identical, use the `ExternalDNSName` attribute for the server instance to define the server's external DNS name. Outside the firewall the `ExternalDNSName` should translate to external IP address of the server. If clients are accessing WebLogic Server over the default channel and `t3`, do not set the `ExternalDNSName` attribute, even if the internal and external DNS names of a WebLogic Server instance are not identical.

Localhost Considerations

If you identify a server instance's listen address as localhost, non-local processes will not be able to connect to the server instance. Only processes on the machine that hosts the server instance will be able to connect to the server instance. If the server instance must be

accessible as localhost (for instance, if you have administrative scripts that connect to localhost), and must also be accessible by remote processes, leave the listen address blank. The server instance will determine the address of the machine and listen on it.

Assigning Names to WebLogic Server Resources

Ensure that each configurable resource in your WebLogic Server environment has a unique name. Each domain, server, machine, cluster, data source, virtual host, or other resource must have a unique name.

Administration Server Address and Port

Identify the DNS name or IP address and listen port of the Administration Server you will use for the cluster.

The Administration Server is the WebLogic Server instance used to configure and manage all the Managed Servers in its domain. When you start a Managed Server, you identify the host and port of its Administration Server.

Managed Server Addresses and Listen Ports

Identify the DNS name or IP address of each Managed Server planned for your cluster.

Each Managed Server in a cluster must have a unique combination of address and listen port number. Clustered server instances on a single non-multihomed machine can have the same address, but must use a different listen port.

Cluster Multicast Address and Port

Identify the address and port you will dedicate to multicast communications for your cluster. A multicast address is an IP address between 224.0.0.0 and 239.255.255.255.

> **Note**
>
> The default multicast value used by WebLogic Server is 239.192.0.0. You should not use any multicast address with the value x.0.0.1.

Server instances in a cluster communicate with each other using multicast—they use multicast to announce their services, and to issue periodic heartbeats that indicate continued availability.

The multicast address for a cluster should not be used for any purpose other than cluster communications. If the machine where the cluster multicast address exists hosts or is accessed by cluster-external programs that use multicast communication, ensure that those multicast communications use a different port than the cluster multicast port. For more information, see Using IP Multicast.

Multicast and Multiple Clusters

Multiple clusters on a network may share a multicast address and multicast port combination if necessary.

Multicast and Multitier Clusters

If you are setting up the recommended multitier architecture, described in Cluster Architectures, with a firewall between the clusters, you will need two dedicated multicast addresses: one for the presentation (servlet) cluster and one for the object cluster. Using two multicast addresses ensures that the firewall does not interfere with cluster communication.

Cluster Address

In WebLogic Server cluster, the *cluster address* is used in stateless beans to construct the host name portion of request URLs.

You can explicitly define the cluster address when you configure a cluster; otherwise, WebLogic Server dynamically generates the cluster address for each new request. For system administration, it is simple to allow the WebLogic Server to generate the cluster address dynamically and is suitable for both development and production environments.

Dynamic Cluster Address

If you do not explicitly define a cluster address when you configure a cluster, and when a clustered server instance receives a remote request, WebLogic Server generates the cluster address, in the form:

```
listenaddress1:listenport1,listenaddress2:listenport2;listenaddress3:
listenport3
```

Each `listen address:listen port` combination in the cluster address corresponds to Managed Server and network channel that received the request.

- If the request was received on the Managed Server's default channel, the `listen address:listen port` combinations in the cluster address reflect the `ListenAddress` and `ListenPort` values from the associated `ServerMBean` and `SSLMBean` instances. See The Default Network Channel in *Administering Server Environments for Oracle WebLogic Server*.
- If the request was received on a custom network channel, the `listen address:listen port` in the cluster address reflect the `ListenAddress` and `ListenPort` values from `NetworkAccessPointMBean` that defines the channel. For more information about network channels in a cluster, see Configuring Network Channels For a Cluster in *Administering Server Environments for Oracle WebLogic Server*.

The number of `ListenAddress:ListenPort` combinations included in the cluster address is governed by the value of the `NumberOfServersInClusterAddress` attribute on the `ClusterMBean`, which is 3 by default.

> **Note**
>
> You can set `NumberOfServersInClusterAddress` attribute to a minimum value of 1. This attribute is used when WebLogic Server dynamically constructs cluster address for use with EJBs. In WebLogic Server cluster, the cluster address is used in entity and stateless beans to construct the host name portion of requested URLs. You can explicitly define the cluster address while configuring a cluster. Otherwise, WebLogic Server dynamically generates the cluster address for each new request. Allowing WebLogic Server to dynamically generate the cluster address is the simplest and convenient method for System Administration, and both development and production environments.

You can modify the value of **Number Of Servers In Cluster Address** on the **Environment**: **Clusters**: *myCluster*: **General** page of the WebLogic Remote Console.

- If there are *fewer* Managed Servers available in the cluster than the value of `NumberOfServersInClusterAddress`, the dynamically generated cluster address contains a `ListenAddress:ListenPort` combination for each of the running Managed Servers.
- If there are *more* Managed Servers available in the cluster than the value of `NumberOfServersInClusterAddress`, WebLogic Server randomly selects a subset of the available instances that is equal to the value of `NumberOfServersInClusterAddress`, and uses the `ListenAddress:ListenPort` combination for those instances to form the cluster address.

The order in which the `ListenAddress:ListenPort` combinations appear in the cluster address is random; from request to request, the order will vary.

Explicitly Defining Cluster Address for Production Environments

If you explicitly define a cluster address for a cluster in a production environment, specify the cluster address as a DNS name that maps to the IP addresses or DNS names of each WebLogic Server instance in the cluster.

If you define the cluster address as a DNS name, the listen ports for the cluster members are not specified in the cluster address—it is assumed that each Managed Server in the cluster has the same listen port number. Because each server instance in a cluster must have a unique combination of address and listen port, if a cluster address is a DNS name, each server instance in the cluster must have:

- A unique address and
- The same listen port number

When clients obtain an initial JNDI context by supplying the cluster DNS name, `weblogic.jndi.WLInitialContextFactory` obtains the list of all addresses that are mapped to the DNS name. This list is cached by WebLogic Server instances, and new initial context requests are fulfilled using addresses in the cached list with a round-robin algorithm. If a server instance in the cached list is unavailable, it is removed from the list. The address list is refreshed from the DNS service only if the server instance is unable to reach any address in its cache.

Using a cached list of addresses avoids certain problems with relying on DNS round-robin alone. For example, DNS round-robin continues using all addresses that have been mapped to the domain name, regardless of whether or not the addresses are reachable. By caching the address list, WebLogic Server can remove addresses that are unreachable, so that connection failures aren't repeated with new initial context requests.

> **Note**
>
> The Administration Server should not participate in a cluster. Ensure that the Administration Server's IP address is not included in the cluster-wide DNS name. See Administration Server Considerations.

Explicitly Defining Cluster Address for Development and Test Environments

If you explicitly define a cluster address for using it in development environments, you can use a cluster DNS name for the cluster address, as described in Explicitly Defining Cluster Address for Production Environments.

Alternatively, you can define the cluster address as a list that contains the DNS name (or IP address) and listen port of each Managed Server in the cluster, as shown in the examples below:

```
DNSName1:port1,DNSName1:port2,DNSName1:port3
IPaddress1:port1,IPaddress2:port2;IPaddress3:port3
```

> **Note**
>
> Each cluster member has a unique address and port combination.

Explicitly Defining Cluster Address for Single, Multihomed Machine

If your cluster runs on a single, multihomed machine, and each server instance in the cluster uses a different IP address, define the cluster address using a DNS name that maps to the IP addresses of the server instances in the cluster. If you define the cluster address as a DNS name, specify the same listen port number for each of the Managed Servers in the cluster.

Cluster Implementation Procedures

The following sections describe how to get a clustered application up and running from the installation of the WebLogic Server through the initial deployment of application components.

Configuration Roadmap

This section lists typical cluster implementation tasks, and highlights key configuration considerations. The exact process you follow is driven by the unique characteristics of your environment and the nature of your application. The following list describes the cluster implementation tasks:

1. Install WebLogic Server
2. Create a Clustered Domain
3. Configure Node Manager
4. Configure Load Balancing Method for EJBs and RMIs
5. Configure Server Affinity for Distributed JMS Destinations
6. Configuring Load Balancers that Support Passive Cookie Persistence

7. Configure Proxy Plug-Ins
8. Configure Replication Groups
9. Configure Migratable Targets for Pinned Services
10. Package Applications for Deployment
11. Deploy Applications
12. Deploying_ Activating_ and Migrating Migratable Services
13. Configure In-Memory HTTP Replication
14. Additional Configuration Topics

Not every step is required for every cluster implementation. Additional steps may be necessary in some cases.

Install WebLogic Server

If you have not installed WebLogic Server prior, install it. For more information about WebLogic Server installation, see Planning the Oracle WebLogic Server Installation in *Installing and Configuring Oracle WebLogic Server and Coherence*.

- If the cluster will run on a single machine, do a single installation of WebLogic Server under the `/Oracle` directory to use for all clustered instances.
- For remote, networked machines, install the same version of WebLogic Server on each machine. Each machine:
  - Must have permanently assigned, static IP addresses. You cannot use dynamically-assigned IP addresses in a clustering environment.
  - Must be accessible to clients. If the server instances are behind a firewall and the clients are in front of the firewall, each server instance must have a public IP address that can be reached by the clients.
  - Must be located on the same LAN and must be reachable via IP multicast.

Create a Clustered Domain

The are multiple methods of creating a clustered domain. See Methods of Configuring Clusters.

For instructions to create a cluster using the:

- Configuration Wizard, do the following:
  1. For instructions on creating a domain, see Creating a WebLogic Domain in *Creating WebLogic Domains Using the Configuration Wizard*.
  2. For instructions on configuring a cluster, see Clusters in *Creating WebLogic Domains Using the Configuration Wizard*.
- See Create and Configure Clusters in the *Oracle WebLogic Remote Console Online Help*.

Starting a WebLogic Server Cluster

There are multiple methods of starting a cluster. The available options include the command-line interface, scripts that contain the necessary commands, and Node Manager.

> **Note**
>
> Node Manager eases the process of starting servers, and restarting them after failure.
>
> To use Node Manager, you must first configure a Node Manager process on each machine that hosts Managed Servers in the cluster. See Configure Node Manager.

Regardless of the method you use to start a cluster, start the Administration Server first, then start the Managed Servers in the cluster.

Follow the instructions below to start the cluster from a command shell.

> **Note**
>
> Each server instance is started in a separate command shell.

1. Open a command shell.
2. Change directory to the domain directory that you created with the Configuration Wizard.
3. Type the `StartWebLogic` command to start the Administration Server.
4. Enter user name and password for the domain when prompted.

   The command shell displays messages that report the status of the startup process.
5. Open another command shell so that you can start a Managed Server.
6. Change directory to the domain directory that you created with the Configuration Wizard.
7. Type the following command:

   `StartManagedWebLogic` *`server_name address:port`*
   where:

   server_name is the name of the Managed Server you wish to start

   address is the IP address or DNS name for the Administration Server for the domain

   port is the listen port for the Administration Server for the domain
8. Enter user name and password for the domain when prompted.

   The command shell displays messages that report the status of the startup process.

   > **Note**
   >
   > After you start a Managed Server, it listens for heartbeats from other running server instances in the cluster. The Managed Server builds its local copy of the cluster-wide JNDI tree, as described in How WebLogic Server Updates the JNDI Tree, and displays status messages when it has synchronized with each running Managed Server in the cluster. The synchronization process can take a minute or so.
9. To start another server instance in the cluster, return to step 5 and continue through step 8.
10. When you have started all Managed Servers in the cluster, the cluster startup process is complete.

Configure Node Manager

Node Manager is a standalone program provided with WebLogic Server that is useful for starting a Managed Server that resides on a different machine than its Administration Server. Node Manager also provides features that help increase the availability of Managed Servers in your cluster. For instructions to configure and use Node Manager, see *Administering Node Manager for Oracle WebLogic Server*.

Configure Load Balancing Method for EJBs and RMIs

Follow the instructions in this section to select the load balancing algorithm for EJBs and RMI objects.

Unless you explicitly specify otherwise, WebLogic Server uses the round-robin algorithm as the default load balancing strategy for clustered object stubs. To understand alternative load balancing algorithms, see Load Balancing for EJBs and RMI Objects.

Using the WebLogic Remote Console to change the default load balancing algorithm, do as follows:

1. In the **Edit Tree**, go to **Environment**, then **Clusters**.
2. In the table, select the name of your cluster.
3. Enter the desired load balancing algorithm in the **Default Load Algorithm** field.
4. Select **Show Advanced Fields**.
5. Enter the desired value in the **Service Age Threshold** field.
6. Click **Save**.

Specifying a Timeout Value For RMIs

You can enable a timeout option when making calls to the ReplicationManager by setting the `ReplicationTimeoutEnabled` in the ClusterMBean to `true`.

The timeout value is equal to the multicast heartbeat timeout. Although you can customize the multicast timeout value, the ReplicationManager timeout cannot be changed. This restriction exists because the ReplicationManager timeout does not affect cluster membership. A missing multicast heartbeat causes the member to be removed from the cluster and the timed out ReplicationManager call will choose a new secondary server to connect to.

> **Note**
>
> It is possible that a cluster member will continue to send multicast heartbeats, but will be unable to process replication requests. This could potentially cause an uneven distribution of secondary servers. When this situation occurs, a warning message is recorded in the server logs.

Configure Server Affinity for Distributed JMS Destinations

To understand the server affinity support provided by WebLogic Server for JMS, see Load Balancing for JMS.

Configuring Load Balancers that Support Passive Cookie Persistence

Load balancers that support passive cookie persistence can use information from the WebLogic Server session cookie to associate a client with the WebLogic Server instance that hosts the session. The session cookie contains a string that the load balancer uses to identify the primary server instance for the session.

For a discussion of external load balancers, session cookie persistence, and the WebLogic Server session cookie, see Load Balancing HTTP Sessions with an External Load Balancer.

To configure the load balancer to work with your cluster, use the facilities of the load balancer to define the offset and length of the string constant.

Assuming that the Session ID portion of the session cookie is the default length of 52 bytes, on the load balancer, set:

- string offset to 53 bytes, the default Random Session ID length plus 1 byte for the delimiter character.
- string length to 10 bytes.

If your application or environmental requirements dictate that you change the length of the Random Session ID from its default value of 52 bytes, set the string offset on the load balancer accordingly. The string offset must equal the length of the Session ID plus 1 byte for the delimiter character.

> **Note**
>
> For vendor-specific instructions for configuring Big-IP load balancers, see Configuring BIG-IP Hardware with Clusters.

Configure Proxy Plug-Ins

Refer to the instructions in this section if you wish to load balance servlets and JSPs using a proxy plug-in. A proxy plug-in proxies requests from a Web server to WebLogic Server instances in a cluster, and provides load balancing and failover for the proxied HTTP requests.

For information about load balancing using proxy plug-ins, see Load Balancing with a Proxy Plug-in. For information about connection and failover using proxy plug-ins, see Replication and Failover for Servlets and JSPs , and Accessing Clustered Servlets and JSPs Using a Proxy.

- If you use WebLogic Server as a Web server, set up `HttpClusterServlet` using the instructions in Set Up the HttpClusterServlet.
- If you use a supported third-party Web server, set up a product-specific plug-in (for a list of supported Web servers, see Load Balancing with a Proxy Plug-in) follow the instructions in *Using Oracle WebLogic Server Proxy Plug-Ins*.

> **Note**
>
> Each Web server that proxies requests to a cluster must have an identically configured plug-in.

Set Up the HttpClusterServlet

To use the HTTP cluster servlet, configure it as the default Web application on your proxy server machine, as described in the steps below. For an introduction to Web applications, see Understanding Web Applications, Servlets, and JSPs in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

1. If you have not already done so, configure a separate, non-clustered Managed Server to host the HTTP Cluster Servlet.
2. Create the `web.xml` deployment descriptor file for the servlet. This file must reside in the `\WEB-INF` subdirectory of the Web application directory. A sample deployment descriptor for the proxy servlet is provided in Sample web.xml.
   a. Define the name and class for the servlet in the `<servlet>` element in `web.xml`. The servlet name is `HttpClusterServlet`. The servlet class is `weblogic.servlet.proxy.HttpClusterServlet`.
   b. Identify the clustered server instances to which the proxy servlet will direct requests in the `<servlet>` element in `web.xml`, by defining the `WebLogicCluster` parameter.
   c. Optionally, define the following `<KeyStore>` initialization parameters to use two-way SSL with your own identity certificate and key. If no `<KeyStore>` is specified in the deployment descriptor, the proxy will assume one-way SSL.
      - `<KeyStore>`: The key store location in your Web application.
      - `<KeyStoreType>`: The key store type. If it is not defined, the default type will be used instead.
      - `<PrivateKeyAlias>`: The private key alias.
      - `<KeyStorePasswordProperties>`: A property file in your Web application that defines encrypted passwords to access the key store and private key alias. The file contents looks like this:

        ```
        KeyStorePassword={AES}yWv/i0qhfM4/IvzoghzjHj/xpJUkQPF8OWuSfh0f0Ss=
        PrivateKeyPassword={AES}wr86u9Z5DHr+5p7WIbzTDSy4M/sl7EYnX/K5xzcarDQ=
        ```

        You must use the `weblogic.security.Encrypt` command-line utility to encrypt the password. For more information on the `Encrypt` utility, as well as the `CertGen`, and `der2pem` utilities, see Using the Oracle WebLogic Server Java Utilities in the *Command Reference for Oracle WebLogic Server*.
   d. Create `<servlet-mapping>` stanzas to specify the requests that the servlet will proxy to the cluster, using the `<url-pattern>` element to identify specific file extensions, for example `*.jsp`, or `*.html`. Define each pattern in a separate `<servlet-mapping>` stanza.

      You can set the `<url-pattern>` to "`/`" to proxy any request that cannot be resolved by WebLogic Server to the remote server instance. If you do so, you must also specifically map the following extensions: `*.jsp`, `*.htm`, and `*.html`, to proxy files ending with those extensions. For an example, see Sample web.xml.
   e. You can enable the `WLProxyPassThrough` attribute to allow the header to be passed through a chain of proxies and the `WLProxySSLPassThrough` attribute so that the use of SSL is passed on to WebLogic Server. For a complete description of these attributes, see General Parameters for Web Server Plug-Ins in *Using Oracle WebLogic Server Proxy Plug-Ins*.

f. Define, as appropriate, any additional parameters. See Table 9-1 for a list of key parameters. See Parameters for Web Server Plug-ins in *Using Oracle WebLogic Server Proxy Plug-Ins* for a complete list. Follow the syntax instructions in Proxy Servlet Deployment Parameters.

3. Create the `weblogic.xml` deployment descriptor file for the servlet. This file must reside in the `\WEB-INF` subdirectory of the Web application directory.

   Assign the proxy servlet as the default Web application for the Managed Server on the proxy machine by setting the `<context-root>` element to a forward slash character (/) in the `<weblogic-web-app>` stanza. For an example, see Sample weblogic.xml.

4. In the WebLogic Remote Console, deploy the servlet to the Managed Server on your proxy server machine. See Deploying Applications in the *Oracle WebLogic Remote Console Online Help*.

Sample web.xml

This section contains a sample deployment descriptor file (`web.xml`) for `HttpClusterServlet`.

`web.xml` defines parameters that specify the location and behavior of both versions of the proxy servlet.

- The `DOCTYPE` stanza specifies the document type definition (DTD) used by WebLogic Server to validate `web.xml`.
- The `servlet` stanza:
  - Specifies the location of the proxy plug-in servlet class. The file is located in the `weblogic.jar` in your *`WL_HOME`*`/server/lib` directory. You do not have to specify the servlet's full directory path in `web.xml` because `weblogic.jar` is put in your `CLASSPATH` when you start WebLogic Server.
  - Identifies the host name (either DNS name or IP address) and listen port of each Managed Servers in the cluster, using the `WebLogicCluster` parameter.
  - Identifies the key store initialization parameters to use two-way SSL with your own identity certificate and key.
- The three `servlet-mapping` stanzas specify that the servlet will proxy URLs that end in '/', 'htm', 'html', or 'jsp' to the cluster.

For parameter definitions, see Proxy Servlet Deployment Parameters.

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
         http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"version="3.1">
<servlet>
  <servlet-name>HttpClusterServlet</servlet-name>
    <servlet-class>
      weblogic.servlet.proxy.HttpClusterServlet
    </servlet-class>
  <init-param>
    <param-name>WebLogicCluster</param-name>
    <param-value>hostname1:7736|hostname2:7736|hostname:7736</param-value>
  </init-param>
  <init-param>
    <param-name>KeyStore</param-name>
    <param-value>/mykeystore</param-value>
  </init-param>
  <init-param>
    <param-name>KeyStoreType</param-name>
```

```
    <param-value>jks</param-value>
  </init-param>
  <init-param>
    <param-name>PrivateKeyAlias</param-name>
    <param-value>passalias</param-value>
  </init-param>
  <init-param>
    <param-name>KeyStorePasswordProperties</param-name>
    <param-value>mykeystore.properties</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>*.html</url-pattern>
</servlet-mapping>
</web-app>
```

Sample weblogic.xml

This section contains a sample `weblogic.xml` file. The `<context-root>` deployment parameter is set to "/". This makes the proxy servlet the default Web application for the proxy server.

```
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web Application 9.1//EN"
"http://www.bea.com/servers/wls810/dtd/weblogic
810-web-jar.dtd">
  <weblogic-web-app>
    <context-root>/</context-root>
  </weblogic-web-app>
```

Proxy Servlet Deployment Parameters

Key parameters for configuring the behavior of the proxy servlet in `web.xml` are listed in Before You Start.

The parameters for the proxy servlet are the same as those used to configure WebLogic Server plug-ins for Apache and Microsoft Web servers. For a complete list of parameters for configuring the proxy servlet and the plug-ins for third-part Web servers, see Parameters for Web Server Plug-ins in *Using Oracle WebLogic Server Proxy Plug-Ins*.

The syntax for specifying the parameters, and the file where they are specified, is different for the proxy servlet and for each of the plug-ins.

For the proxy servlet, specify the parameters in `web.xml`, each in its own `<init-param>` stanza within the `<servlet>` stanza of `web.xml`. For example:

```
<init-param>
    <param-name>ParameterName</param-name>
    <param-value>ParameterValue</param-value>
</init-param>
```

Table 9-1 Proxy Servlet Deployment Parameter

Parameter	Usage

`WebLogicCluster`	`<init-param>`
	`<param-name>WebLogicCluster</param-name>`
	`<param-value>WLS1.com:`*`port`*`|WLS2.com:`*`port`*
	`</param-value>`
	
	Where `WLS1.com` and `WLS2.com` are the host names of servers in the cluster, and *port* is a port where the host is listening for HTTP requests.
	If you are using SSL between the plug-in and WebLogic Server, set the port number to the SSL listen port (see Specify Listen Ports) and set the `SecureProxy` parameter to ON.

`SecureProxy`	`<init-param>`
	`<param-name>SecureProxy</param-name>`
	`<param-value>`*`ParameterValue`*`</param-value>`
	`</init-param>`
	
	Valid values are ON and OFF.
	If you are using SSL between the plug-in and WebLogic Server, set the port number to the SSL listen port (see Specify Listen Addresses) and set the `SecureProxy` parameter to ON.

`DebugConfigInfo`	`<init-param>`
	`<param-name>DebugConfigInfo</param-name>`
	`<param-value>`*`ParameterValue`*`</param-value>`
	`</init-param>`
	
	Valid values are ON and OFF.
	If set to ON, you can query the `HttpClusterServlet` for debugging information by adding a request parameter of `?__WebLogicBridgeConfig` to any request. (Note: There are two underscore (_) characters after the ?.) For security reasons, it is recommended that you set the `DebugConfigInfo` parameter to OFF in a production environment.

`ConnectRetrySecs`	Interval in seconds that the servlet will sleep between attempts to connect to a server instance. Assign a value less than `ConnectTimeoutSecs`.
	The number of connection attempts the servlet makes before returning an `HTTP 503/Service Unavailable` response to the client is `ConnectTimeoutSecs` divided by `ConnectRetrySecs`.
	Syntax:
	
	`<init-param>`
	`<param-name>ConnectRetrySecs</param-name>`
	`<param-value>`*`ParameterValue`*`</param-value>`
	`</init-param>`

Table 9-1 (Cont.) Proxy Servlet Deployment Parameter

Parameter	Usage

`ConnectTimeoutSecs`	Maximum time in seconds that the servlet will attempt to connect to a server instance. Assign a value greater than `ConnectRetrySecs`.
	If `ConnectTimeoutSecs` expires before a successful connection, an `HTTP 503/Service Unavailable` response is sent to the client.
	Syntax:
	`<init-param>`
	`<param-name>ConnectTimeoutSecs</param-name>`
	`<param-value>`*`ParameterValue`*`</param-value>`
	`</init-param>`

`PathTrim`	String trimmed by the plug-in from the beginning of the original URL, before the request is forwarded to the cluster.
	Syntax:
	`<init-param>`
	`<param-name>PathTrim</param-name>`
	`<param-value>`*`ParameterValue`*`</param-value>`
	`</init-param>`
	Example:
	If the URL
	`http://myWeb.example.com/weblogic/foo`
	is passed to the plug-in for parsing and if `PathTrim` has been set to
	`/weblogic`
	the URL forwarded to WebLogic Server is:
	`http://myWeb.example.com:7001/foo`

`TrimExt`	The file extension to be trimmed from the end of the URL.
	Syntax:
	`<init-param>`
	`<param-name>TrimExt</param-name>`
	`<param-value>`*`ParameterValue`*`</param-value>`
	`</init-param>`

Table 9-1 (Cont.) Proxy Servlet Deployment Parameter

Parameter	Usage
`clientCertProxy`	Specifies to trust client certificates in the `WL-Proxy-Client-Cert` header. Valid values are true and false. The default value is false. This setting is useful if user authentication is performed on the proxy server—setting `clientCertProxy` to true causes the proxy server to pass on the certs to the cluster in a special header, `WL-Proxy-Client-Cert`. The `WL-Proxy-Client-Cert` header can be used by any client with direct access to WebLogic Server. WebLogic Server takes the certificate information from that header, trusting that is came from a secure source (the plug-in) and uses that information to authenticate the user. For this reason, if you set `clientCertProxy` to true, use a connection filter to ensure that WebLogic Server accepts connections only from the machine on which the plug-in is running. See Using Network Connection Filters in *Developing Applications with the WebLogic Security Service*.

`PathPrepend`	String that the servlet prepends to the original URL, after `PathTrim` is trimmed, before forwarding the URL to the cluster.
	
	`<init-param>`
	`<param-name>PathPrepend</param-name>`
	`<param-value>`*`ParameterValue`*`</param-value>`
	`</init-param>`

`RoutingHandlerClassName`	Extends the proxy servlet to support Web service cluster routing. See Managing Web Services in a Cluster in *Developing JAX-WS Web Services for Oracle WebLogic Server*.
	
	`<init-param>`
	`<param-name>RoutingHandlerClassName</param-name>`
	`<param-value>`
	`weblogic.wsee.jaxws.cluster.proxy.SOAPRoutingHandler`
	`</param-value>`
	`</init-param>`

Accessing Applications Via the Proxy Server

Ensure that application clients access through the proxy server are deployed to your cluster. Address client requests to the listen address and listen port of the proxy server.

If you have problems:

- Ensure that all server instances have unique address/port combinations.

  Each of the server instances in the configuration must have a unique combination of listen address and listen port.
- Ensure that the proxy servlet is the default application for the proxy server.

  If you get "page not found" error when you try to your application, make sure that `weblogic.xml` is in `\WEB-INF` for the application and that it sets the `context-root` deployment parameter to "/".
- When all else fails, restart

If you are having problems, try to restart all your servers. Some of the changes you made while configuring your setup may not persist to the configuration file.

- Verify Your Configuration

  To verify the configuration of the `HttpClusterServlet`:

  1. Set the `DebugConfigInfo` parameter in `web.xml` to ON.
  2. Use a Web browser to access the following URL:

     ```
     http://myServer:port/placeholder.jsp?__WebLogicBridgeConfig
     ```

  Where:

  – `myServer` is the Managed Server on the proxy machine where `HttpClusterServlet` runs,

  – `port` is the port number on the server that is listening for HTTP requests, and

  – `placeholder.jsp` is a file that does not exist on the server.

  The plug-in gathers configuration information and run-time statistics and returns the information to the browser. For more information, see Parameters for Web Server Plug-ins in *Using Oracle WebLogic Server Proxy Plug-Ins.*

Configure Replication Groups

To support automatic failover for servlets and JSPs, WebLogic Server replicates HTTP session states in memory. You can further control where secondary states are placed using *replication groups*. A replication group is a preferred list of clustered instances to be used for storing session state replicas.

If your cluster will host servlets or stateful session EJBs, you may want to create replication groups of WebLogic Server instances to host the session state replicas.

For instructions on how to determine which server instances should participate in each replication group, and to determine each server instance's preferred replication group, follow the instructions in Using Replication Groups.

Using the WebLogic Remote Console, follow these steps to configure replication groups for each WebLogic Server instance:

1. In the **Edit Tree**, go to **Environment**, then **Servers**.
2. In the table, select the name of the server you want to configure.
3. Select the **Cluster** tab.
4. Enter values for the following attribute fields:
   - **Replication Group**: Enter the name of the replication group to which this server instance belongs.
   - **Preferred Secondary Group**: Enter the name of the replication group you would like to use to host replicated HTTP session states for this server instance.
5. Click **Save**.

Configure Migratable Targets for Pinned Services

WebLogic Server enables you to configure an optional migratable target, which is a special target that can migrate from one server in a cluster to another. As such, a migratable target provides a way to group pinned services that should move together. When the migratable

target is migrated, all services hosted by that target are migrated. Pinned services include JMS-related services (for example, JMS servers, SAF agents, path services, and persistent stores) or the JTA Transaction Recovery Service.

If you want to use a migratable target, configure the target server list before deploying or activating the service in the cluster. If you do not configure a migratable target in the cluster, migratable services can be migrated to any available WebLogic Server instance in the cluster. See Understanding Migratable Targets In a Cluster.

Package Applications for Deployment

You must package applications before you deploy them to WebLogic Server. For more information ,see Packaging Applications Using wlpackage in *Developing Applications for Oracle WebLogic Server*.

Deploy Applications

Clustered objects in WebLogic Server should be deployed homogeneously. To ensure homogeneous deployment, when you select a target, use the cluster name rather than individual WebLogic Server instances in the cluster. If cluster is dynamic or mixed cluster type, you can select singleton or distributed deployment based on your need for one single instance across entire cluster or one instance per cluster member.

The Console automates deploying replica-aware objects to clusters. When you deploy an application or object to a cluster, the Console automatically deploys it to all members of the cluster (whether they are local to the Administration Server machine or they reside on remote machines). For a discussion of application deployment in clustered environments, see Methods of Configuring Clusters. For a broad discussion of deployment topics, see *Deploying Applications to Oracle WebLogic Server*.

> **Note**
>
> All server instances in your cluster should be running when you deploy applications to the cluster using the WebLogic Remote Console.

Deploying to a Single Server Instance (Pinned Deployment)

Deploying an application to a server instance, rather than all cluster members is called a pinned deployment. Although a pinned deployment targets a specific server instance, all server instances in the cluster must be running during the deployment process.

You can perform a pinned deployment using the command line `weblogic.Deployer`.

Pinned Deployment from the Command Line

From a command shell, use the following syntax to target a server instance:

```
java weblogic.Deployer -activate -name ArchivedEarJar -source C:/MyApps/JarEar.ear -
target server1
```

Cancelling Cluster Deployments

You can cancel a deployment using the command line `weblogic.Deployer`.

Cancel Deployment from the Command Line

From a command shell, use the following syntax to cancel the deployment task ID:

```
java weblogic.Deployer -adminurl http://admin:7001 -cancel -id tag
```

Viewing Deployed Applications

To view a deployed application in the WebLogic Remote Console:

1. In the **Edit Tree**, go to **Deployments**, then **App Deployments**.
2. View a list of deployed applications in the table.

Undeploying Deployed Applications

To undeploy a deployed application using the WebLogic Remote Console, do as follows:

1. In the **Monitoring Tree**, go to **Deployments**, then **Application Management**.
2. In the table, select the check box next to the application you want to stop (undeploy).
3. Click **Stop** and choose one of:
   - **When work completes**: to allow the application to finish its work and for all currently connected users to disconnect.
   - **Force stop now**: to stop the application immediately, regardless of the work that is being performed and the users that are connected.
   - **Servicing non-administration requests**: to stop the application after all its work has finished, but to then put the application in Administration Mode so it remains accessible for administrative purposes.

Deploying, Activating, and Migrating Migratable Services

The sections that follow provide guidelines and instructions for deploying, activating, and migrating migratable services.

Deploying JMS to a Migratable Target Server Instance

The migratable target that you create defines the scope of server instances in the cluster that can potentially host a migratable service. You must deploy or activate a pinned service on one of the server instances listed in the migratable target to migrate the service within the target server list at a later time. Use the instructions that follow to deploy a JMS service on a migratable target, or activate the JTA transaction recovery system so that you can migrate it later.

> **Note**
>
> If you did not configure a migratable target, deploy the JMS server to any WebLogic Server instance in the cluster; you can then migrate the JMS server to any other server instance in the cluster (no migratable target is used).

Before you begin, use the instructions in Configure Migratable Targets for Pinned Services, to create a migratable target for the cluster. Then, deploy JMS-related services to a migratable target.

Using the Remote Console, in the **Edit Tree**, go to **Services**: **JMS System Resources**, and select *myJMSService* for which you want to deploy a migratable target. Then select the **Target** tab and, from the **Target** drop-down list, select the migratable target where you want to deploy the JMS-related service.

Activating JTA as a Migratable Service

The JTA recovery service is automatically started on one of the server instances listed in the migratable target for the cluster; you do not have to deploy the service to a selected server instance.

If you did not configure a JTA migratable target, WebLogic Server activates the service on any available WebLogic Server instance in the cluster.

Configure In-Memory HTTP Replication

To support automatic failover for servlets and JSPs, WebLogic Server replicates HTTP session states in memory.

> **Note**
>
> WebLogic Server can also maintain the HTTP session state of a servlet or JSP using file-based or JDBC-based persistence. For more information on these persistence mechanisms, see Using Sessions and Session Persistence in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

In-memory HTTP session state replication is controlled separately for each application you deploy. For the application, the parameter that controls it is `PersistentStoreType`, which appears within the session-descriptor element in the WebLogic deployment descriptor file and `weblogic.xml`.

```
domain_directory/applications/application_directory/WEB-INF/weblogic.xml
```

To use in-memory HTTP session state replication across server instances in a cluster, set the `PersistentStoreType` to `replicated`. The fragment below shows the appropriate XML from `weblogic.xml`.

```
<session-descriptor>
<persistent-store-type>replicated</persistent-store-type>
</session-descriptor>
```

Additional Configuration Topics

The sections below contain useful tips for particular cluster configurations.

Configure IP Sockets

For best socket performance, Oracle recommends that you use the native socket reader implementation, rather than the pure-Java implementation, on machines that host WebLogic Server instances.

If you must have to use the pure-Java socket reader implementation for host machines, you can still improve the performance of socket communication by configuring the proper number of socket reader threads for each server instance and client machine.

- To learn more about how IP sockets are used in a cluster, and why native socket reader threads provide best performance, see Peer-to-Peer Communication Using IP Sockets, and Client Communication via Sockets.
- For instructions to determine the number of socket reader threads necessary for your cluster, see Determining Potential Socket Usage in *Tuning Performance of Oracle WebLogic Server*. If you are deploying a servlet cluster in a multitier cluster architecture, this has an effect on how many sockets you require, as described in Configuration Considerations for Multitier Architecture.

The following sections have instructions to configure native socket reader threads for host machines and set the number of reader threads for host and client machines.

Configure Native IP Sockets Readers on Machines that Host Server Instances

To configure a WebLogic Server instance to use the native socket reader threads implementation:

1. Using the WebLogic Remote Console, in the **Edit Tree**, go to **Environment**, then **Servers**, then *myServer*.
2. Select the name of the server instance you want to configure.
3. Click the **Advanced** tab, then the **Tuning** subtab.
4. Turn on the **Enable Native IO** option.
5. Click **Save**.

Set the Number of Reader Threads on Machines that Host Server Instances

By default, a WebLogic Server instance creates three socket reader threads upon starting. If you determine that your cluster system may utilize more than three sockets during peak periods, increase the number of socket reader threads by using the following instructions:

1. Using the WebLogic Remote Console, in the **Edit Tree**, go to **Environment**, then **Servers**, then *myServer*.
2. Select the name of the server instance you want to configure.
3. Click the **Advanced** tab, then the **Tuning** subtab.
4. Edit the percentage of Java reader threads in the **JavaSocketMuxer Socket Readers** field. The number of Java socket readers is computed as a percentage of the number of total execute threads.
5. Click **Save**.

Set the Number of Reader Threads on Client Machines

On client machines, you can configure the number of socket reader threads in the Java Virtual Machine (JVM) that runs the client. Specify the socket readers by defining the `-Dweblogic.ThreadPoolSize=`*`value`* and `-Dweblogic.ThreadPoolPercentSocketReaders=`*`value`* options in the Java command line for the client.

Configure Multicast Time-To-Live (TTL)

If your cluster spans multiple subnets in a WAN, the value of the Multicast TTL parameter for the cluster must be high enough to ensure that routers do not discard multicast packets before they reach their final destination. The Multicast TTL parameter sets the number of network hops a multicast message makes before the packet can be discarded. Configuring the Multicast TTL parameter appropriately reduces the risk of losing the multicast messages that are transmitted among server instances in the cluster.

For more information about planning your network topology to ensure that multicast messages are reliably transmitted, see If Your Cluster Spans Multiple Subnets In a WAN.

In the WebLogic Remote Console, to configure the Multicast TTL for a cluster, change the **Multicast TTL** value in the **Edit Tree**, on the **Environment**: **Clusters**: *myCluster*: **Messaging** page, click **Show Advanced Fields**. The following `config.xml` excerpt shows a cluster with a Multicast TTL value of three. This value ensures that the cluster's multicast messages can pass through three routers before being discarded:

```
<Cluster
    Name="testcluster"
    ClusterAddress="wanclust"
    MulticastAddress="wanclust-multi"
    MulticastTTL="3"
/>
```

> **Note**
>
> When relying upon the Multicast TTL value, it is important to remember that within a clustered environment it is possible that timestamps across servers may not always be synchronized. For example, this can occur in replicated HTTP sessions and EJBs.
>
> When the ClusterDebug flag is enabled and cluster members clocks are not synchronized, an error is printed to the server log.

Configure Multicast Buffer Size

If multicast storms occur because server instances in a cluster are not processing incoming messages on a timely basis, you can increase the size of multicast buffers. For information on multicast storms, see If Multicast Storms Occur.

TCP/IP kernel parameters can be configured with the UNIX `ndd` utility. The `udp_max_buf` parameter controls the size of send and receive buffers (in bytes) for a UDP socket. The appropriate value for `udp_max_buf` varies from deployment to deployment. If you are experiencing multicast storms, increase the value of `udp_max_buf` by 32K, and evaluate the effect of this change.

Do not change `udp_max_buf` unless necessary. Before changing `udp_max_buf`, read the warning in the UDP Parameter With Additional Caution in *Solaris Tunable Parameters Reference Manual*.

Configure Multicast Data Encryption

WebLogic Server allows you to encrypt multicast messages that are sent between clusters. You can enable this option in the WebLogic Remote Console with the following instructions:

1. In the **Edit Tree**, go to **Environment**: **Clusters**: *myCluster*.
2. Click the **Messaging** tab and then **Show Advanced Fields**.
3. Turn on the **Enable Multicast Data Encryption** option.

Only the data portion of the multicast message is encrypted. Information contained in the multicast header is not encrypted.

Configure Machine Names

Configure a machine name, if:

- Your cluster will span multiple machines, and multiple server instances will run on individual machines in the cluster.

  Or

- You plan to run Node Manager on a machine that does not host an Administration Server.

WebLogic Server uses configured machine names to determine whether or not two server instances reside on the same physical hardware. Machine names are generally used with machines that host multiple server instances. If you do not define machine names for such installations, each instance is treated as if it resides on separate physical hardware. This can negatively affect the selection of server instances to host secondary HTTP session state replicas, as described in Using Replication Groups.

Configuration Notes for Multitier Architecture

If your cluster has a multitier architecture, see the configuration guidelines in Configuration Considerations for Multitier Architecture.

Enable URL Rewriting

In its default configuration, WebLogic Server uses client-side cookies to keep track of the primary and secondary server instance that host the client's servlet session state. If client browsers have disabled cookie usage, WebLogic Server can also keep track of primary and secondary server instances using URL rewriting. With URL rewriting, both locations of the client session state are embedded into the URLs passed between the client and proxy server. To support this feature, you must ensure that URL rewriting is enabled on the WebLogic Server cluster. For instructions on how to enable URL rewriting, see Using URL Rewriting Instead of Cookies in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

10

Dynamic Clusters

Learn about dynamic clusters and how to create, configure, and use dynamic clusters in Oracle WebLogic Server.

This chapter includes the following sections:

What Are Dynamic Clusters?

Dynamic clusters consist of server instances that can be dynamically scaled up to meet the resource needs of your application. A dynamic cluster uses a single server template to define configuration for a specified number of generated (dynamic) server instances.

When you create a dynamic cluster, the dynamic servers are preconfigured and automatically generated for you, enabling you to easily scale up the number of server instances in your dynamic cluster when you need additional server capacity. You can start the dynamic servers without manual configuration and adding them to the cluster.

If you need additional server instances on top of the number you originally specified, you can increase the maximum number of servers instances (dynamic) in the dynamic cluster configuration or manually add configured server instances to the dynamic cluster. A dynamic cluster that contains both dynamic and configured server instances is called a mixed cluster.

The following table defines the terminology associated with dynamic clusters:

Term	Definition
dynamic cluster	A cluster that contains one or more generated (dynamic) server instances that are based on a single shared server template.
configured cluster	A cluster in which you manually configure and add each server instance.
dynamic server	A server instance that is generated by WebLogic Server when creating a dynamic cluster. Configuration is based on a shared server template.
configured server	A server instance for which you manually configure attributes.
mixed cluster	A cluster that contains both dynamic and configured server instances.
server template	A prototype server definition that contains common, non-default settings and attributes that can be assigned to a set of server instances, which then inherit the template configuration. For dynamic clusters, the server template is used to generate the dynamic servers. See Server Templates in *Understanding Domain Configuration for Oracle WebLogic Server*.

Term	Definition
dynamic cluster size	A specified number of server instances allowed to be created in a dynamic cluster. These server instances are dynamically added to the configuration at runtime and associated `ServerLifeCycleRuntimeMBeans` are created. **Note** When you expand the dynamic cluster, the size must not exceed the limit set in Maximum Dynamic Cluster Size. WebLogic Server adds only running dynamic server instances up to the Maximum Dynamic Cluster Size limit. Also, when you shrink the dynamic cluster, the number of running dynamic servers cannot be below the limit set in Minimum Dynamic Cluster Size.

You cannot configure dynamic servers individually; there are no server instance definitions in the `config.xml` file when using a dynamic cluster. Therefore, you cannot override the server template with server-specific attributes or target applications to an individual dynamic server instance. Example 10-1 shows an example `config.xml` file that includes a dynamic cluster.

Example 10-1 Example config.xml File Using a Dynamic Cluster

```
<server-template>
    <name>dynamic-cluster-server-template</name>
    <accept-backlog>2000</accept-backlog>
    <auto-restart>true</auto-restart>
    <restart-max>10</restart-max>
    <startup-timeout>600</startup-timeout>
</server-template>

<cluster>
    <name>dynamic-cluster</name>
    <cluster-messaging-mode>unicast<cluster-messaging-mode>
    <dynamic-servers>
      <server-template>dynamic-cluster-server-template</server-template>
      <calculated-listen-ports>true</calculated-listen-ports>
      <calculated-machine-names>true</calculated-machine-names>
      <server-name-prefix>dynamic-server-</server-name-prefix>
      <dynamic-cluster-size>4</dynamic-cluster-size>
      <max-dynamic-cluster-size>8</max-dynamic-cluster-size>
    </dynamic-servers>
</cluster>
```

Why Do You Use Dynamic Clusters?

With dynamic clusters, you can scale up your cluster when you need additional server capacity by starting one or more of the preconfigured dynamic server instances. You do not need to manually configure a new server instance and add it to the cluster or perform a system restart.

How Do Dynamic Clusters Work?

The following sections describes to create, configure, and use dynamic clusters.

Creating and Configuring Dynamic Clusters

To create a dynamic cluster, perform the following actions:

- Specify the number of server instances you anticipate at peak load.
- Create or select the server template upon which you want to base server configuration.
- Define how WebLogic Server should calculate server-specific attributes.

WebLogic Server then generates the specified number of dynamic server instances and applies the calculated attribute values to each dynamic server instance.

> **Note**
>
> Ensure you have the performance capacity to handle the maximum number of server instances you specify in the dynamic cluster configuration. For information on design and deployment best practices when creating a cluster, see Clustering Best Practices.

You can create dynamic clusters using WLST and the WebLogic Remote Console.

Example 10-2 demonstrates using WLST. For information about using the WebLogic Remote Console, see Create a Dynamic Cluster in the *Oracle WebLogic Remote Console Online Help*. When creating a dynamic cluster in either console, WebLogic Server creates the server template, dynamic cluster, and specified number of server instances for you. You do not have to specify them individually. You can configure dynamic clusters using any of the administration tools listed in Summary of System Administration Tools and APIs in *Understanding Oracle WebLogic Server*.

Using Server Templates

Server templates define common configuration attributes that a set of server instances share. Dynamic clusters use server templates for dynamic server configuration.

For more information, see Server Templates in *Understanding Domain Configuration for Oracle WebLogic Server*.

Calculating Server-Specific Attributes

You cannot configure individual dynamic server instances or override values in the server template at the dynamic server level when using a dynamic cluster. Server-specific attributes, such as server name, machines, and listen ports, must be calculated using the information provided when creating the dynamic cluster.

> **Note**
>
> You must set a unique Listen Address value for the Managed Server instance that will host the JTA Transaction Recovery service. Otherwise, the migration will fail.

WebLogic Server calculates and applies the following server-specific attributes using the instance ID of the dynamic server:

- Server name
- (Optional) Listen ports (clear text and SSL)
- (Optional) Network access point listen ports
- (Optional) Machines or virtual machines

For more information, see the following sections:

Calculating Server Names

The calculated server name is controlled by the `ServerNamePrefix` attribute. Server names are the specified prefix followed by the index number, starting with the value specified by the `serverNameStartingIndex` attribute (which defaults to 1). For example, if the prefix is set to `dyn-server-` and the starting index is set to `4`, then the dynamic servers will have the names `dyn-server-4`, `dyn-server-5`, and so on for the number of server instances you specified.

Calculating Listen Ports

The settings in the dynamic cluster configuration or server template determine the listen ports for the server instances in your dynamic cluster. If you do not calculate listen ports when creating your dynamic cluster, WebLogic Server uses the value in the server template. If you do not define listen ports in the dynamic cluster configuration or server template, WebLogic Server uses the default value. Listen port settings are controlled by the `CalculatedListenPorts` attribute. For more information about these settings, see Create a Dynamic Cluster in the *Oracle WebLogic Remote Console Online Help*.

If you explicitly define a base listen port for your dynamic cluster in the server template or the dynamic cluster configuration, the listen port value for the first dynamic server instance is the base listen port incremented by one. For each additional dynamic server instance, the listen port value is incremented by one. If the default base listen port is used, WebLogic Server increments the "hundreds" digit by one and continues from there for each dynamic server instance. Table 10-1 shows examples of calculated listen port values.

Table 10-1 Calculating Listen Ports

Listen Port Type	Configuration Setting in Server Template	Dynamic Server Listen Port Values

Server listen port	Base listen port not set	`dyn-server-1: 7101`
		`dyn-server-2: 7102`
		`...`

Server listen port	Base listen port set to 7300	`dyn-server-1: 7301`
		`dyn-server-2: 7302`
		`...`

Server SSL listen port	SSL base listen port not set	`dyn-server-1: 8101`
		`dyn-server-2: 8102`
		`...`

Server SSL listen port	SSL base listen port set to 8200	`dyn-server-1: 8201`
		`dyn-server-2: 8202`
		`...`

Table 10-1 (Cont.) Calculating Listen Ports

Listen Port Type	Configuration Setting in Server Template	Dynamic Server Listen Port Values

Server network access point listen port	Network access point listen port not set	`dyn-server-1: 9101`
		`dyn-server-2: 9102`
		`...`

Server network access point listen port	Network access point listen port set to 9200	`dyn-server-1: 9201`
		`dyn-server-2: 9202`
		`...`

Server replication ports	Replication ports set to 8100	`dyn-server-1: 8100`
		`dyn-server-2: 8101`
		`...`

Server replication ports	Replication ports set to 8100-8104	`dyn-server-1: 8100-8104`
		`dyn-server-2: 8105-8109`
		`dyn-server-3: 8110-8114`
		`...`

You can override listen ports at server startup by using system properties. For example:

To override the listen port: `-Dweblogic.ListenPort=7305`

To override the SSL listen port: `-Dweblogic.ssl.ListenPort=7403`

To override the listen port of the network access point named `mynap`: `-Dweblogic.networkaccesspoint.mynap.ListenPort=8201`

Calculating Machine Names

The dynamic cluster attributes `CalculatedMachineNames` and `MachineNameMatchExpression` control how server instances in a dynamic cluster are assigned to a machine. If the `CalculatedMachineNames` attribute is set to `false`, then the dynamic servers will not be assigned to a machine. If the `CalculatedMachineNames` attribute is set to `true`, then the `MachineNameMatchExpression` attribute is used to select the set of machines used for the dynamic servers. If the `MachineNameMatchExpression` attribute is not set, then all of the machines in the domain are selected. Assignments are made using a round robin algorithm. Table 10-2 shows examples of machine assignments in a dynamic cluster.

Table 10-2 Calculating Machine Names

Machines in Domain	`MachineNameMatchExpression` Configuration	Dynamic Server Machine Assignments

M1, M2	Not set	`dyn-server-1: M1`
		`dyn-server-2: M2`
		`dyn-server-3: M1`
		`...`

Table 10-2 (Cont.) Calculating Machine Names

Machines in Domain	MachineNameMatchExpression Configuration	Dynamic Server Machine Assignments

Ma1, Ma2, Mb1, Mb2	Ma1, Mb*	`dyn-server-1: Ma1`
		`dyn-server-2: Mb1`
		`dyn-server-3: Mb2`
		`dyn-server-4: Ma1`
		`...`

Starting and Stopping Servers in Dynamic Clusters

To easily manage server startup and shutdown in dynamic clusters, use the WLST `scaleUp` and `scaleDown` commands.

The `scaleUp` command increases the number of running servers for the specified dynamic cluster. The non-running server instance with the lowest server ID starts first, followed by the next highest non-running server ID, until the specified number of server instances is started.

You can start one, all, or any number of server instances in the dynamic cluster by specifying the desired number with the `numServers` argument in the `scaleUp` command. If all available server instances are already running, the `scaleUp` command increases the size of the cluster to the minimum number of requested server instances before starting the specified number of servers. For more information about expanding cluster size, see Expanding or Reducing Dynamic Clusters.

The `scaleDown` command gracefully shuts down the specified number of running servers. The server instance with the highest server ID shuts down first, followed by the next highest ID, until the specified number of server instances is shut down. For more information, see scaleUp and scaleDown in *WLST Command Reference for Oracle WebLogic Server*.

You can also start and stop server instances in dynamic clusters using the same methods you use to start and stop server instances in configured clusters: the WebLogic Remote Console, WLST `start` and `shutdown` commands, Node Manager, and start scripts. Depending on which startup method you choose and the tasks you have already performed, you may have to follow several other procedures before you can start server instances. See Starting and Stopping Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

> **Note**
>
> Before you begin, ensure that WebLogic Server is installed on all hosts where you want to run your server instances. If you want to use Node Manager to start and stop your server instances, then you must also run Node Manager on these hosts.

Expanding or Reducing Dynamic Clusters

When you create a dynamic cluster, WebLogic Server generates the number of dynamic servers you specify. Before you decide upon the number of server instances, ensure that you have the performance capacity to handle the desired number.

The dynamic server instances are based on the configuration you specified in the server template and calculated attributes.

- To expand your cluster, start any number of the preconfigured dynamic servers.
- To shrink your dynamic cluster, shut down the excess number of dynamic servers.

If you need additional server capacity on top of the number of server instances you originally specified, increase the maximum number of dynamic servers in the dynamic cluster configuration. To reduce the number of server instances in the dynamic cluster, decrease the value of the maximum number of dynamic servers attribute. Before lowering this value, shut down the server instances you plan to remove.

> **Note**
>
> When you expand the cluster, the size must not exceed the limit set in Maximum Dynamic Cluster Size. WebLogic Server adds only running dynamic server instances up to the Maximum Dynamic Cluster Size limit. Also, when you shrink the cluster, the number of running dynamic servers cannot be below the limit set in Minimum Dynamic Cluster Size.

You can also use the WLST `scaleUp` and `scaleDown` commands to manage your dynamic cluster. To increase the number of dynamic servers in the dynamic cluster, use the scaleUp command and enable the `updateConfiguration` argument. WLST will increase the size of the cluster by the specified number of servers and start the server instances.

> **Note**
>
> The `scaleDown` command lowers the cluster dynamic cluster size setting when its `updateConfiguration` option is enabled, which in turn can abandon JMS persistent messages and JTA transactions that are associated with retired servers. If you are using JMS or JTA, do not use `scaleDown` with its `updateConfiguration` option enabled. See Best Practices for Using Cluster Targeted JMS Services in *Administering JMS Resources for Oracle WebLogic Server*.

To decrease the size of the dynamic cluster, use the `scaleDown` command and enable the `updateConfiguration` argument. WLST will gracefully shut down the specified number of running server instances and remove them from the dynamic cluster. See `scaleUp` and `scaleDown` in *WLST Command Reference for Oracle WebLogic Server*.

> **Note**
>
> You can only use the WLST `scaleUp` and `scaleDown` commands with dynamic server instances. In a mixed cluster, containing both manually configured and dynamic server instances, the `scaleUp` and `scaleDown` commands ignore the configured servers. You must manually start and stop configured server instances in a mixed cluster.
>
> For example, a cluster contains two running dynamic servers and two non-running configured servers. If you use the `scaleUp` command, WLST adds one additional dynamic server instance to your cluster and starts the dynamic server.

The WLST `scaleUp` and `scaleDown` commands provide ways to manually scale your dynamic cluster. For automatic scaling, you can configure elasticity for your dynamic cluster. Elasticity enables a dynamic cluster to perform scaling and re-provisioning operations automatically in response to demand or on a calendar based schedule. WebLogic Server provides elasticity for dynamic clusters through the Policies and Actions system of the WebLogic Diagnostic Framework (WLDF). See *Configuring Elasticity in Dynamic Clusters for Oracle WebLogic Server*.

Using Whole Server Migration with Dynamic Clusters

WebLogic Server supports whole server migration with dynamic and mixed clusters. While configuration differs depending on the cluster type, whole server migration behavior is the same for all clusters. For information on how to enable whole server migration for dynamic clusters, see Whole Server Migration with Dynamic and Mixed Clusters.

Deploying Applications to Dynamic Clusters

When deploying applications to a dynamic cluster, you must target the application to the entire cluster. You cannot target an application to an individual server instance because dynamic clusters do not have individual dynamic server configuration. When you deploy an application to the dynamic cluster, all servers in the cluster, both dynamic and static, will deploy the application.

To deploy an application to a dynamic cluster, follow the same process as deploying to configured clusters. See Deploy Applications and Application Deployment for Clustered Configurations.

For a broad discussion of deployment topics, see *Deploying Applications to Oracle WebLogic Server*.

Using WebLogic Web Server Plug-Ins with Dynamic Clusters

Dynamic clusters provide the same WebLogic Web server plug-in support as configured clusters. By default, a Web server plug-in uses the `DynamicServerList` parameter to receive information about cluster changes, such as new server instances in a configured or dynamic cluster. Upon recognizing a cluster membership change, the plug-in automatically updates its server list.

For general information about using Web server plug-ins with WebLogic Server, see *Using Oracle WebLogic Server Proxy Plug-Ins*. For more information about the `DynamicServerList` parameter or the `WebLogicCluster` parameter (required when proxying to a WebLogic Server cluster), see General Parameters for Web Server Plug-Ins in *Using Oracle WebLogic Server Proxy Plug-Ins*.

Limitations and Considerations When Using Dynamic Clusters

When using dynamic clusters with WebLogic Server, be aware of certain limitations and considerations.

- You cannot override values in the server template at the individual dynamic server level because there are no individual server definitions in the `config.xml` file when using dynamic clusters.
- You need to ensure that the values are set on server template, such that each Dynamic server inherits unique values for required parameters. For example, set the value to

`LLR_DYN_SRV_${id}` for LLR table under Server Template configuration. `${id}` would be replaced with server number at runtime for each dynamic server.

- Ensure that you have the performance capacity to handle the maximum number of server instances you specify in the dynamic cluster configuration.
- Dynamic clusters do not support targeting to any individual dynamic server instance. Therefore, the following cannot be used with dynamic clusters:
  - Deployments that cannot target to a cluster, including migratable targets. Therefore, you cannot create a user-defined migratable target for a dynamic cluster.

    > **Note**
    >
    > This limitation does not apply to the internal migratable targets for JTA or JMS.

  - Configuration attributes that refer to individual servers. This includes migratable target attributes such as constrained candidate servers, user preferred server, all candidate servers and hosting server. By default, all members of the dynamic cluster are candidate servers for migratable target.
  - Configuration attributes that are server specific. This includes replication groups, preferred secondary groups, and candidate machines (server level).
  - Constrained candidates for singleton services. You cannot limit a singleton service to dynamic servers.
- For whole server migration with a dynamic cluster, you cannot limit the list of candidate machines that the dynamic cluster specifies, as the server template does not list candidate machines.
- Dynamic clusters also have JMS limitations. See Simplified JMS Cluster Configuration in *Administering JMS Resources for Oracle WebLogic Server*.
- A dynamic server with multiple network channels configured cannot have those channels listen on different interfaces.
- The `scaleDown` command lowers the cluster dynamic cluster size setting when its `updateConfiguration` option is enabled, which in turn can abandon JMS persistent messages and JTA transactions that are associated with retired servers. If you are using JMS or JTA, do not use `scaleDown` with its `updateConfiguration` option enabled. See Best Practices for Using Cluster Targeted JMS Services in *Administering JMS Resources for Oracle WebLogic Server*.

Dynamic Clusters Example

Examine the WLST commands to create a dynamic cluster.

Example 10-2 includes inline comments and describes how to:

1. Create a server template and specify the desired server attributes in the server template.
2. Create a dynamic cluster and specify the desired cluster attributes.
3. Set the server template for the dynamic cluster.
4. Set the maximum number of server instances you want in the dynamic cluster.
5. Start and stop the server instances in the dynamic cluster.

Example 10-2 Creating Dynamic Clusters with WLST

```
#
# This example demonstrates the WLST commands needed to create a dynamic cluster
# (dynamic-cluster). The dynamic cluster uses a server template
# dynamic-cluster-server-template. To keep this example simple, error handling
# was omitted.
#
connect()
edit()
startEdit()
#
# Create the server template for the dynamic servers and set the attributes for
# the dynamic servers. Setting the cluster is not required.
#
dynamicServerTemplate=cmo.createServerTemplate("dynamic-cluster-server-template")
dynamicServerTemplate.setAcceptBacklog(2000)
dynamicServerTemplate.setAutoRestart(true)
dynamicServerTemplate.setRestartMax(10)
dynamicServerTemplate.setStartupTimeout(600)
#
# Create the dynamic cluster, set the number of dynamic servers, and designate the
server template.
#
dynCluster=cmo.createCluster("dynamic-cluster")
dynServers=dynCluster.getDynamicServers()
dynServers.setDynamicClusterSize(4)
dynServers.setServerTemplate(dynamicServerTemplate)
#
# Dynamic server names will be dynamic-server-1, dynamic-server-2, dynamic-server-3,
# dynamic-server-4.
#
dynServers.setServerNamePrefix("dynamic-server-")
#
# Listen ports and machines assignments will be calculated. Using a round-robin
# algorithm, servers will be assigned to machines with names that start with
# dyn-machine.
#
dynServers.setCalculatedMachineNames(true)
dynServers.setMachineNameMatchExpression("dyn-machine*")
#
# activate the changes
#
activate()
```

The resulting `config.xml` file is:

```
<server-template>
    <name>dynamic-cluster-server-template</name>
    <accept-backlog>2000</accept-backlog>
    <auto-restart>true</auto-restart>
    <restart-max>10</restart-max>
    <startup-timeout>600</startup-timeout>
</server-template>

<cluster>
    <name>dynamic-cluster</name>
    <cluster-messaging-mode>unicast<cluster-messaging-mode>
    <dynamic-servers>
      <server-template>dynamic-cluster-server-template</server-template>
      <calculated-listen-ports>true</calculated-listen-ports>
      <calculated-machine-names>true</calculated-machine-names>
      <server-name-prefix>dynamic-server-</server-name-prefix>
```

```
      <dynamic-cluster-size>4</dynamic-cluster-size>
      <max-dynamic-cluster-size>8</max-dynamic-cluster-size>
    </dynamic-servers>
</cluster>
```

11

Configuring and Managing Coherence Clusters

Learn how to define Coherence clusters in an OracleWebLogic Server domain and associate an Oracle Coherence cluster with multiple Oracle WebLogic Server clusters.

This chapter includes the following sections:

Overview of Coherence Clusters

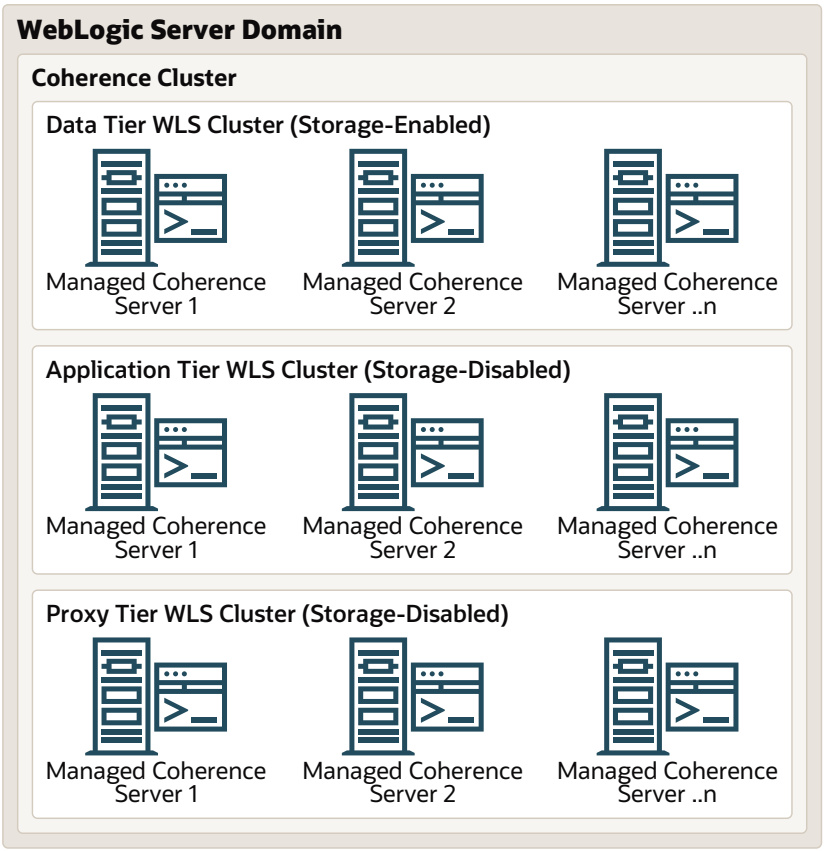
Coherence clusters consist of multiple Managed Coherence server instances that distribute data in-memory to increase application scalability, availability, and performance. An application interacts with the data in a local cache and the distribution and backup of the data is performed automatically across cluster members.

Coherence clusters are different than WebLogic Server clusters. They use different clustering protocols and are configured separately. Multiple WebLogic Server clusters can be associated with a Coherence cluster and a WebLogic Server domain typically contains a single Coherence cluster. Managed servers configured as Coherence cluster members are referred to as managed Coherence servers. For more information, see Configuring Coherence in the *Oracle WebLogic Remote Console Online Help*.

Managed Coherence servers can be explicitly associated with a Coherence cluster or they can be associated with a WebLogic Server cluster that is associated with a Coherence cluster. Managed Coherence servers are typically set up in tiers that are based on their type: a data tier for storing data, an application tier for hosting applications, and a proxy tier that allows external clients to access caches. See Coherence Deployment Tiers in the *Oracle WebLogic Remote Console Online Help*.

Figure 11-1 shows a conceptual view of a Coherence cluster in a WebLogic Server domain.

Figure 11-1 Conceptual View of a Coherence Domain Topology



Setting Up a Coherence Cluster

A WebLogic Server domain typically contains a single Coherence cluster. The cluster is represented as a single system-level resource (`CoherenceClusterSystemResource`). A `CoherenceClusterSystemResource` instance is created using the WebLogic Remote Console or WLST.

A Coherence cluster can contain any number of managed Coherence servers. The servers can be standalone managed servers or can be part of a WebLogic Server cluster that is associated with a Coherence cluster. Typically, multiple WebLogic Server clusters are associated with a Coherence cluster. For details on creating WebLogic Server clusters for use by Coherence, see Creating Coherence Deployment Tiers.

> **Note**
>
> Cloning a managed Coherence server does not clone its association with a Coherence cluster. The managed server will not be a member of the Coherence cluster. You must manually associate the cloned managed server with the Coherence cluster.

For information on setting up Coherence clusters, see Configure Coherence: Main Steps in the *Oracle WebLogic Remote Console Online Help*.

Define a Coherence Cluster Resource

Use the WebLogic Remote Console to define a Coherence cluster resource. See Create a Coherence Cluster in the *Oracle WebLogic Remote Console Online Help*.

Create Standalone Managed Coherence Servers

Managed Coherence servers are managed server instances that are associated with a Coherence cluster. Managed Coherence servers join together to form a Coherence cluster and are often referred to as cluster members. Cluster members have seniority and the senior member performs cluster tasks (for example, issuing the cluster heart beat).

> **Note**
>
> - Managed Coherence servers and standalone Coherence cluster members (those that are not managed within a WebLogic Server domain) can join the same cluster. However, standalone cluster members cannot be managed from within a WebLogic Server domain; operational configuration and application life cycles must be manually administered and monitored.
> - Standalone Coherence cluster members must be configured to use Well Known Addresses (WKA) when joining a Coherence cluster that is managed in a WebLogic Server domain.
> - The Administration Server is typically not used as a managed Coherence server in a production environment.

Managed Coherence servers are distinguished by their role in the cluster. A best practice is to use different managed server instances (and preferably different WebLogic Server clusters) for each cluster role.

- **Storage-enabled**: A managed Coherence server that is responsible for storing data in the cluster. Coherence applications are packaged as Grid ARchives (GAR) and deployed on storage-enabled managed Coherence servers.
- **Storage-disabled**: A managed Coherence server that is not responsible for storing data and is used to host Coherence applications (cache clients). A Coherence application GAR is packaged within an enterprise archive (EAR) and deployed on storage-disabled managed Coherence servers.
- **Proxy**: A managed Coherence server that is storage-disabled and allows external clients (non-cluster members) to use a cache. A Coherence application GAR is deployed on managed Coherence proxy servers.

Use the WebLogic Remote Console to create managed Coherence servers. See Create a Managed Coherence Server in the *Oracle WebLogic Remote Console Online Help*.

Creating Coherence Deployment Tiers

Coherence supports different topologies within a WebLogic Server domain to provide varying levels of performance, scalability, and ease of use.

For example, during development, a single standalone managed server instance may be used as both a cache server and a cache client. The single-server topology is easy to set up and

use, but does not provide optimal performance or scalability. For production, Coherence is typically set up using WebLogic Server clusters. A WebLogic Server cluster is used as a Coherence data tier and hosts one or more cache servers; a different WebLogic Server cluster is used as a Coherence application tier and hosts one or more cache clients; and (if required) different WebLogic Server clusters are used for the Coherence proxy tier that hosts one or more managed Coherence proxy servers and the Coherence extend client tier that hosts extend clients. The tiered topology approach provides optimal scalability and performance.

For more information, see Coherence Deployment Tiers in the *Oracle WebLogic Remote Console Online Help*.

The instructions in this section use both the **Coherence Clusters** and **Servers** pages in the WebLogic Remote Console to create Coherence deployment tiers. WebLogic Server clusters and managed servers instances can be associated with a Coherence cluster resource using the `ClusterMBean` and `ServerMBean` respectively. Managed servers that are associated with a WebLogic Server cluster inherit the cluster's Coherence settings. However, the settings may not be reflected in the **Servers** settings page.

Configuring and Managing a Coherence Data Tier

A Coherence data tier is a WebLogic Server cluster that is associated with a Coherence cluster and hosts any number of storage-enabled managed Coherence servers. Managed Coherence servers in the data tier store and distribute data (both primary and backup) on the cluster. The number of managed Coherence servers that are required in a data tier depends on the expected amount of data that is stored in the Coherence cluster and the amount of memory available on each server. In addition, a cluster must contain a minimum of four physical computers to avoid the possibility of data loss during a computer failure.

Coherence artifacts (such as Coherence configuration files, POF serialization classes, filters, entry processors, and aggregators) are packaged as a GAR and deployed on the data tier. For details on packaging and deploying Coherence applications, see *Developing Oracle Coherence Applications for Oracle WebLogic Server*. For details on calculating cache size and hardware requirements, see the production checklist in *Administering Oracle Coherence*.

Create a Coherence Data Tier

Use the WebLogic Remote Console to create a Coherence data tier. See Create a Coherence Data Tier in the *Oracle WebLogic Remote Console Online Help*.

Create Managed Coherence Servers for a Data Tier

Use the WebLogic Remote Console to create Managed Servers for a Coherence data tier.

1. Create a WebLogic Server Managed Server, as described in Create a Managed Server.
2. On the Managed Server that you just created, use the **Cluster** drop-down list to select a cluster that is working as a data tier WebLogic Server cluster. The Managed Server inherits the Coherence settings from the WebLogic Server cluster data tier.
3. Click **Save**.
4. Repeat these steps to create additional Managed Servers as required.

Configuring and Managing a Coherence Application Tier

A Coherence Application tier is a WebLogic Server cluster that is associated with a Coherence cluster and hosts any number of storage-disabled managed Coherence servers. Managed

Coherence servers in the application tier host applications (cache factory clients) and are Coherence cluster members. Multiple application tiers can be created for different applications.

Clients in the application tier are deployed as EARs and implemented using Jakarta EE standards such as servlet, JSP, and EJB. Coherence artifacts (such as Coherence configuration files, POF serialization classes, filters, entry processors, and aggregators) must be packaged as a GAR and also deployed within an EAR. For details on packaging and deploying Coherence applications, see *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

Create a Coherence Application Tier

Use the WebLogic Remote Console to create a Coherence application tier. See Create a Coherence Application Tier in the *Oracle WebLogic Remote Console Online Help*.

Create Managed Coherence Servers for an Application Tier

Use the WebLogic Remote Console to create Managed Servers for a Coherence application tier.

1. Create a WebLogic Server Managed Server, as described in Create a Managed Server.
2. On the Managed Server that you just created, use the **Cluster** drop-down list to select a cluster that is working as an application tier WebLogic Server cluster. The Managed Server inherits the Coherence settings from the application tier WebLogic Server cluster.
3. Click **Save**.
4. Repeat these steps to create additional Managed Servers as required.

Configuring and Managing a Coherence Proxy Tier

A Coherence proxy tier is a WebLogic Server cluster that is associated with a Coherence cluster and hosts any number of managed Coherence proxy servers. Managed Coherence proxy servers allow Coherence*Extend clients to use Coherence caches without being cluster members. The number of managed Coherence proxy servers that are required in a proxy tier depends on the number of expected clients. At least two proxy servers must be created to allow for load balancing; however, additional servers may be required when supporting a large number of client connections and requests.

For details on Coherence*Extend and creating extend clients, see *Developing Remote Clients for Oracle Coherence*.

Create a Coherence Proxy Tier

Use the WebLogic Remote Console to create a Coherence proxy tier. See Create a Coherence Proxy Tier in the *Oracle WebLogic Remote Console Online Help*.

Create Managed Coherence Servers for a Proxy Tier

Use the WebLogic Remote Console to create Managed Servers for a Coherence proxy tier.

1. Create a WebLogic Server Managed Server, as described in Create a Managed Server.
2. On the Managed Server that you just created, use the **Cluster** drop-down list to select a cluster that is working as a proxy tier WebLogic Server cluster. The Managed Server inherits the Coherence settings from the proxy tier WebLogic Server cluster.

3. Click **Save**.
4. Repeat these steps to create additional Managed Servers as required.

Configure Coherence Proxy Services

Coherence proxy services are clustered services that manage remote connections from extend clients. Proxy services are defined and configured in a `coherence-cache-config.xml` file within the `<proxy-scheme>` element. The definition includes, among other settings, the TCP listener address (IP, or DNS name, and port) that is used to accept client connections. For details on the `<proxy-scheme>` element, see *Developing Applications with Oracle Coherence*. Two ways to set up proxy services are:

Using a Name Service

A name service is a specialized listener that allows extend clients to connect to a proxy service by name. Clients connect to the name service, which returns the addresses of all proxy services on the cluster. The naming service provides an efficient setup and is typically preferred in a Coherence proxy tier.

> **Note**
>
> If a domain includes multiple tiers (for example, a data tier, an application tier, and a proxy tier), then the proxy tier should be started first, before a client can connect to the proxy.

A name service automatically starts on port `7574` (the same default port that the Tangosol Cluster Management Protocol (TCMP) socket uses) when a proxy service is configured on a managed Coherence proxy server. The reuse of the same port minimizes the number of ports that are used by Coherence and simplifies firewall configuration.

To configure a proxy service and enable the name service on the default TCMP port:

1. Edit the `coherence-cache-config.xml` file and create a `<proxy-scheme>` definition and do not explicitly define a socket address. The following example defines a proxy service that is named `TcpExtend` and automatically enables a cluster name service. A proxy address and ephemeral port is automatically assigned and registered with the cluster's name service.

```
...
<caching-schemes>
   ...
   <proxy-scheme>
      <service-name>TcpExtend</service-name>
      <autostart>true</autostart>
   </proxy-scheme>
</caching-schemes>
...
```

2. Deploy the `coherence-cache-config.xml` file to each managed Coherence proxy server in the Coherence proxy tier. Typically, the `coherence-cache-config.xml` file is included in a GAR file. However, for the proxy tier, use a cluster cache configuration file to override the `coherence-cache-config.xml` file that is located in the GAR. This allows a single GAR to be deployed to the cluster and the proxy tier. For details on using a cluster cache configuration file, see Overriding a Cache Configuration File.

To connect to a name service, a client's `coherence-cache-config.xml` file must include a `<name-service-addresses>` element, within the `<tcp-initiator>` element, of a remote cache

or remote invocation definition. The <name-service-addresses> element provides the socket address of a name service that is on a managed Coherence proxy server. The following example defines a remote cache definition and specifies a name service listening at host 192.168.1.5 on port 7574. The client automatically connects to the name service and gets a list of all managed Coherence proxy servers that contain a TcpExtend proxy service. The cache on the cluster must also be called TcpExtend. In this example, a single address is provided. A second name service address could be provided in a failure at the primary address. For details on client configuration and proxy service load balancing, see Configuring Extend Proxies in *Developing Remote Clients for Oracle Coherence*.

```
<remote-cache-scheme>
   <scheme-name>extend-dist</scheme-name>
   <service-name>TcpExtend</service-name>
   <initiator-config>
      <tcp-initiator>
         <name-service-addresses>
            <socket-address>
               <address>192.168.1.5</address>
               <port>7574</port>
            </socket-address>
         </name-service-addresses>
      </tcp-initiator>
   </initiator-config>
</remote-cache-scheme>
```

The name service listens on the cluster port (7574) by default and is available on all machines running Coherence cluster nodes. If the target cluster uses the default TCMP cluster port, then the port can be omitted from the configuration.

> **Note**
>
> - The <service-name> value must match the proxy scheme's <service-name> value; otherwise, a <proxy-service-name> element must also be provided in a remote cache and remote invocation scheme that contains the value of the <service-name> element that is configured in the proxy scheme.
> - In previous Coherence releases, the name service automatically listened on a member's unicast port instead of the cluster port.
> - An address provider can also be used to specify name service addresses.

Using an Address Provider

An address provider specifies the TCP listener address (IP, or DNS name, and port) for a proxy service. The listener address can be explicitly defined within a <proxy-scheme> element in a coherence-cache-config.xml file; however, the preferred approach is to define address providers in a cluster configuration file and then reference the addresses within the <proxy-scheme> element. The latter approach decouples deployment configuration from application configuration and allows network addresses to change without having to update a coherence-cache-config.xml file.

To use an address provider:

1. Use the *myCOHcluster*: **Coherence Address Providers** page to create address provider definitions. The CoherenceAddressProvidersBean MBean exposes the address provider definition. An address provider contains a unique name in addition to the listener address

for a proxy service. For example, an address provider called `proxy1` might specify host `192.168.1.5` and port `9099` as the listener address.

See Configure a Coherence Address Provider in the *Oracle WebLogic Remote Console Online Help*.

2. Repeat step 1 and create an address provider definition for each proxy service (at least one for each managed Coherence proxy server).
3. For each managed Coherence proxy server, edit the `coherence-cache-config.xml` file and create a `<proxy-scheme>` definition and reference an address provider definition, by name, in an `<address-provider>` element. The following example defines a proxy service that references an address provider named `proxy1`:

```
...
<caching-schemes>
   <proxy-scheme>
      <service-name>TcpExtend</service-name>
      <acceptor-config>
         <tcp-acceptor>
            <address-provider>proxy1</address-provider>
         </tcp-acceptor>
      </acceptor-config>
      <autostart>true</autostart>
   </proxy-scheme>
</caching-schemes>
...
```

4. Deploy each `coherence-cache-config.xml` file to its respective managed Coherence proxy server. Typically, the `coherence-cache-config.xml` file is included in a GAR file. However, for the proxy tier, use a cluster cache configuration file. The cluster cache configuration file overrides the `coherence-cache-config.xml` file that is located in the GAR. This allows the same GAR to be deployed to all cluster members, but then use unique settings that are specific to a proxy tier. For details on using a cluster cache configuration file, see Overriding a Cache Configuration File.

To connect to a proxy service, a client's `coherence-cache-config.xml` file must include a `<remote-addresses>` element, within the `<tcp-initiator>` element of a remote cache or remote invocation definition, that includes the address provider name. For example:

```
<remote-cache-scheme>
   <scheme-name>extend-dist</scheme-name>
   <service-name>TcpExtend</service-name>
   <initiator-config>
      <tcp-initiator>
         <remote-addresses>
            <address-provider>proxy1</address-provider>
         </remote-addresses>
      </tcp-initiator>
   </initiator-config>
</remote-cache-scheme>
```

Clients can also explicitly specify remote addresses. The following example defines a remote cache definition and specifies a proxy service on host `192.168.1.5` and port `9099`. The client automatically connects to the proxy service and uses a cache on the cluster named `TcpExtend`. In this example, a single address is provided. A second address could be provided in case of a failure at the primary address. For details on client configuration and proxy service load balancing, see Configuring Extend Proxies in *Developing Remote Clients for Oracle Coherence*.

```
<remote-cache-scheme>
   <scheme-name>extend-dist</scheme-name>
```

```
   <service-name>TcpExtend</service-name>
   <initiator-config>
      <tcp-initiator>
         <remote-addresses>
            <socket-address>
               <address>192.168.1.5</address>
               <port>9099</port>
            </socket-address>
         </remote-addresses>
      </tcp-initiator>
   </initiator-config>
</remote-cache-scheme>
```

Configuring a Coherence Cluster

A Coherence cluster resource exposes several cluster settings that can be configured for a specific domain.

Many of the settings use default values that can be changed as required. The following instructions assume that a cluster resource has already been created. For details on creating a cluster resource, see Setting Up a Coherence Cluster. For security details, see *Securing Oracle Coherence*.

Use the **Coherence Clusters**: **General** page to configure cluster communication. The `CoherenceClusterSystemResource` MBean and its associated `CoherenceClusterResource` MBean expose cluster settings. The `CoherenceClusterResource` MBean provides access to multiple MBeans for configuring a Coherence cluster.

> **Note**
>
> WLS configuration take precedence over Coherence system properties. In general, change the Coherence configuration in WLS using WLST or a Coherence cluster configuration file instead of using the system properties.

For more information, see Configure Coherence Clusters in the *Oracle WebLogic Remote Console Online Help*.

Use the following tasks to configure cluster settings:

Adding and Removing Coherence Cluster Members

Any existing managed server instance can be added to a Coherence cluster. In addition, managed Coherence servers can be removed from a cluster. Adding and removing cluster members is available when configuring a Coherence Cluster and is a shortcut that is used instead of explicitly configuring each instance. However, when adding existing managed server instances, default Coherence settings may need to be changed. For details on configuring managed Coherence servers, see Configuring Managed Coherence Servers.

Use the **Members** tab in the **Coherence Cluster** settings to select which managed servers or WebLogic Server clusters are associated with a Coherence cluster. When selecting a WebLogic Server cluster, it is recommended that all the managed servers in the WebLogic Server cluster be associated with a Coherence cluster. A `CoherenceClusterSystemResource` exposes all managed Coherence servers as targets. A `CoherenceMemberConfig` MBean is created for each managed server and exposes the Coherence cluster member parameters.

Setting Advanced Cluster Configuration Options

WebLogic Server MBeans expose a subset of Coherence operational settings that are sufficient for most use cases and are detailed throughout this section. These settings are available natively through the WLST utility and the WebLogic Remote Console. For more advanced use cases, use an external Coherence cluster configuration file (`tangosol-coherence-override.xml`), which provides full control over Coherence operational settings.

> **Note**
>
> The use of an external cluster configuration file is only recommended for operational settings that are not available through the provided MBeans. That is, avoid configuring the same operational settings in both an external cluster configuration file and through the MBeans.

Use the **Coherence Cluster**: **General** settings page to enter the path and name of a cluster configuration file that is located on the Administration Server or use the `CoherenceClusterSystemResource` MBean. For details on using a Coherence cluster configuration file, see Specifying an Operational Configuration File in *Developing Applications with Oracle Coherence*, which also provides usage instructions for each element and a detailed schema reference.

Checking Which Operational Configuration is Used

Coherence generates an operational configuration from WebLogic Server MBeans, a Coherence cluster configuration file (if imported), and Coherence system properties (if set). The result is written to the managed Coherence server log if the system property `weblogic.debug.DebugCoherence=true` is set. If you use the WebLogic start-up scripts, you can use the `JAVA_PROPERTIES` environment variable. For example,

```
export JAVA_PROPERTIES=-Dweblogic.debug.DebugCoherence=true
```

Configure Cluster Communication

Cluster members communicate using the TCMP. The protocol operates independently of the WLS cluster protocol. TCMP is an IP-based protocol for discovering cluster members, managing the cluster, provisioning services, and transmitting data. TCMP can be transmitted over different transport protocols and can use both multicast and unicast. TCMP uses multicast UDP for discovery and TCP for data transmission (using TCP/IP Message Bus (TMB)), by default. If the WKA is configured, then TCMP is transmitted over unicast User Datagram Protocol (UDP) for discovery and Transmission Control Protocol (TCP) for data transmission. If Secure Sockets Layer (SSL) is configured for TCMP, then SSL over TCP is used for both discovery and data transmission. The use of different transport protocols and multicast requires support from the underlying network.

Use the **Coherence Clusters**: **General** page to configure cluster communication. The `CoherenceClusterParamsBean` and `CoherenceClusterWellKnownAddressesBean` MBeans expose the cluster communication parameters.

Changing the Coherence Cluster Mode

Coherence clusters support both unicast and multicast communication. Multicast must be explicitly configured and is not the default option. The use of multicast should be avoided in

environments that do not properly support or allow multicast. The use of unicast disables all multicast transmission and automatically uses the Coherence WKA feature to discover and communicate between cluster members.

For details on using multicast, unicast, and WKA in Coherence, see Setting Up a Cluster in *Developing Applications with Oracle Coherence*.

Selecting Unicast For the Coherence Cluster Mode

To use unicast for cluster communication, select **Unicast** from the **Clustering Mode** drop-down list and enter a cluster port or keep the default port, which is `7574`. From Coherence 12.2.1 onwards, you only need to set the coherence cluster name and all the clusters can use same clusterport. For most clusters, the port does not need to be changed. The only reason to change clusterport is interference with an another application using the port. If a different port is required, then the recommended best practice is to select a value between `1024` and `8999`. See Create a Coherence Cluster in the *Oracle WebLogic Remote Console Online Help*.

> **Note**
>
> The Coherence default cluster port is registered with IANA for the coherence application usage.

Specifying Well Known Address Members

When unicast is enabled, use the **Coherence Cluster Well Known Addresses** page to explicitly configure WKA machine addresses. If no addresses are defined for a cluster, then addresses are automatically assigned. The recommended best practice is to always explicitly specify WKA machine addresses when using unicast. See Specify a Coherence Well Known Address in the *Oracle WebLogic Remote Console Online Help*.

In addition, if a domain contains multiple managed Coherence server that are located on different machines, then at least one non-local WKA machine address must be defined to ensure a Coherence cluster is formed; otherwise, multiple individual clusters are formed on each machine. If the managed Coherence servers are all running on the same machine, then a cluster can be created without specifying a non-local listen address.

> **Note**
>
> WKA machine addresses must be explicitly defined in production environments. In production mode, a managed Coherence server fails to start if WKA machines addresses have not been explicitly defined. Automatically assigned WKA machine addresses is a design time convenience and should only be used during development on a single server.

Selecting Multicast For the Coherence Cluster Mode

To use multicast for cluster communication, select **Multicast** from the **Clustering Mode** drop-down list and enter a cluster port and multicast listen address. The same cluster port can be shared across distinct clusters (as identified by the cluster name), even if the clusters run on the same computer or multicast address. Thus, changing the cluster port is not necessary if the cluster name is being set to a value which is unique to the environment. If a different port is required, then the recommended best practice is to select a value between `1024` and `8999`. See Create a Coherence Cluster in the *Oracle WebLogic Remote Console Online Help*.

Use the TTL field to designate how far multicast packets can travel on a network. The time-to-live value is expressed in terms of how many hops a packet survives; each network interface, router, and managed switch is considered one hop. The TTL value should be set to the lowest integer value that works.

Changing the Coherence Cluster Transport Protocol

The following transport protocols are supported for TCMP.

- User Datagram Protocol (UDP) – UDP is the default TCMP transport protocol and is used for both multicast and unicast communication. If multicast is disabled, all communication is done using UDP unicast.
- Transmission Control Protocol (TCP) – The TCP transport protocol is used in network environments that favor TCP communication. All TCMP communication uses TCP if unicast is enabled. If multicast is enabled, TCP is only used for unicast communication and UDP is used for multicast communication.

> **Note**
>
> Selecting TCP sets both TMB and TCP socket provider used by cluster discovery.

- Secure Sockets Layer (SSL) – The SSL/TCP transport protocol is used in network environments that require highly secure communication between cluster members. SSL is only supported with unicast communication; ensure multicast is disabled when using SSL. The use of SSL requires additional configuration. For details on securing Coherence within WebLogic Server, see *Securing Oracle Coherence*.

The `CoherenceClusterParamsBean` MBean exposes the transport protocol setting through the **Transport** drop-down list with the following options:

- TMB: UDP + TMB (default)
- TCP: TCP + TMB
- UDP: UDP + datagram
- SSL: SSL over TCP + TMBS
- SSLUDP: SSL over TCP + SSL over datagram

These options are a combination of the Coherence Cluster Transport Protocol for cluster service communication and reliable point-to-point data service communication. The following table explains these combinations:

Table 11-1 Transport Types

Transport Type	Description
`TMB: UDP + TMB`	For transport type TMB (TCP/IP Message Bus), the cluster service communication uses UDP and the reliable point-to-point data service communication uses TMB.
`TCP: TCP + TMB`	For transport type TCP, the cluster service communication uses TCP and the reliable point-to-point data service communication uses TMB.
`UDP: UDP + datagram`	For transport type UDP, the cluster service communication uses UDP and the reliable point-to-point data service communication uses datagram.

Table 11-1 (Cont.) Transport Types

Transport Type	Description
`SSL: SSL over TCP + TMBS`	For transport type SSL, the cluster service communication uses SSL over TCP and the reliable point-to-point data service communication uses TMBS.
`SSLUDP: SSL over TCP + SSL over datagram`	For transport type SSLUDP, the cluster service communication uses SSL over TCP and the reliable point-to-point data service communication uses SSL over datagram.

For more information about changing these protocols, see Changing Transport Protocols in *Developing Applications with Oracle Coherence*.

Overriding a Cache Configuration File

A Coherence cache configuration file defines the caches that are used by an application. Typically, a cache configuration file is included in a GAR module. A GAR is deployed to all managed Coherence servers in the data tier and can also be deployed as part of an EAR to the application tier. The GAR ensures that the cache configuration is available on every Oracle Coherence cluster member. However, there are use cases that require a different cache configuration file to be used on specific managed Coherence servers. For example, a proxy tier requires access to all artifacts in the GAR but needs a different cache configuration file that defines the proxy services to start. For more information, see Create a Cluster Cache Configuration in the *Oracle WebLogic Remote Console Online Help*.

A cache configuration file can be associated with WebLogic clusters or managed Coherence servers at runtime. In this case, the cache configuration overrides the cache configuration file that is included in a GAR. You can also omit the cache configuration file from a GAR file and assign it at runtime. To override a cache configuration file at runtime, the cache configuration file must be bound to a JNDI name. The JNDI name is defined using the `override-property` attribute of the `<cache-configuration-ref>` element. The element is located in the `coherence-application.xml` file that is packaged in a GAR file. For details on the `coherence-application.xml` file, see Creating a Coherence Application Deployment Descriptor in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

The following example defines an override property named `cache-config/ExamplesGar` that can be used to override the `META-INF/example-cache-config.xml` cache configuration file in the GAR:

```
...
<cache-configuration-ref override-property="cache-config/ExamplesGar">
   META-INF/example-cache-config.xml</cache-configuration-ref>
...
```

At runtime, use the *myCOHCluster*: **Coherence Cache Configs** page to override a cache configuration file. You must supply the same JNDI name that is defined in the `override-property` attribute. The cache configuration can be located on the administration server or at a URL. In addition, you can choose to import the file to the domain or use it from the specified location. Use the **Targets** tab to specify which Oracle Coherence cluster members use the cache configuration file.

The following WLST (online) example demonstrates how a cluster cache configuration can be overridden using a `CoherenceClusterSystemResource` object.

```
edit()
startEdit()
cd('CoherenceClusterSystemResources/myCoherenceCluster/CoherenceCacheConfigs')
create('ExamplesGar', 'CoherenceCacheConfig')
cd('ExamplesGar')
set('JNDIName', 'ExamplesGar')
cmo.importCacheConfigurationFile('/tmp/cache-config.xml')
cmo.addTarget(getMBean('/Servers/coh_server'))
save()
activate()
```

The WLST example creates a `CoherenceCacheConfig` resource as a child. The script then imports the cache configuration file to the domain and specifies the JNDI name to which the resource binds. The file must be found at the path provided. Lastly, the cache configuration is targeted to a specific server. The ability to target a cache configuration resource to certain servers or WebLogic Server clusters allows the application to load different configurations based on the context of the server (cache servers, cache clients, proxy servers, and so on).

The following example demonstrates how the cache configuration resource can be configured as a URL.

```
edit()
startEdit()
cd('CoherenceClusterSystemResources/myCoherenceCluster/CoherenceCacheConfigs')
create('ExamplesGar', 'CoherenceCacheConfig')
cd('ExamplesGar')
set('JNDIName', 'ExamplesGar')
set('CacheConfigurationFile', 'http://cache.locator/app1/cache-config.xml')
cmo.addTarget(getMBean('/Servers/coh_server'))
save()
activate()
```

Configuring Coherence Logging

Use the WebLogic Remote Console to configure cluster logging. See Configure Coherence Logging in the *Oracle WebLogic Remote Console Online Help*. Alternatively, use the `CoherenceLoggingParamsBean` MBean. For details on WebLogic Server logging, see *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*.

Coherence logging configuration includes:

- Disabling and enabling logging
- Changing the default logger name

  WebLogic Server provides two loggers that can be used for Coherence logging: the default `com.oracle.coherence` logger and the `com.oracle.wls` logger. The `com.oracle.wls` logger is generic and uses the same handler that is configured for WebLogic Server log output. This logger does not allow for Coherence-specific configuration. The `com.oracle.coherence` logger allows Coherence-specific configuration, which includes the use of different handlers for Coherence logs.

  > **Note**
  >
  > If logging is configured through a standard `logging.properties` file, then make sure the file uses the same logger name that is currently configured for Coherence logging.

- Changing the log message format

Add or remove information from a log message. A log message can include static text and parameters that are replaced at run time (for example, `{date}`). For details on supported log message parameters, see Changing the Log Message Format in *Developing Applications with Oracle Coherence*.

> **Note**
>
> The following logger configuration will enable a dynamic Coherence logging:
>
> ```
> <logger-severity>Trace</logger-severity>
> <log-file-severity>Trace</log-file-severity>
> <stdout-severity>Debug</stdout-severity>
> <platform-logger-levels>com.oracle.coherence=FINEST</platform-logger-
> levels>
> ```

Configuring Cache Persistence

Coherence persistence manages the persistence and recovery of Coherence distributed caches. Cached data is persisted so that it can be quickly recovered after a catastrophic failure or after a cluster restart due to planned maintenance. For complete details about Coherence cache persistence, see Persisting Caches in *Administering Oracle Coherence*. For additional details, see Configure Coherence Persistence in the *Oracle WebLogic Remote Console Online Help*.

Use the **Coherence Persistence** tab on the **Coherence Cluster** settings page to enable active persistence and to override the default location where persistence files are stored. The `CoherencePersistenceParamsBean` MBean exposes the persistence parameters. Managed Coherence servers must be restarted for persistence changes to take affect.

On-demand persistence allows a cache service to be manually persisted and recovered upon request (a snapshot) using the persistence coordinator. The persistence coordinator is exposed as an MBean interface (`PersistenceCoordinatorMBean`) that provides operations for creating, archiving, and recovering snapshots of a cache service. To use the MBean, JMX must be enabled on the cluster. For details about enabling JMX management and accessing Coherence MBeans, see Using JMX to Manage Oracle Coherence in *Managing Oracle Coherence*. Active persistence automatically persists cache contents on all mutations and automatically recovers the contents on cluster/service startup. The persistence coordinator can still be used in active persistence mode to perform on-demand snapshots.

Configuring Cache Federation

The federated caching feature federates cache data asynchronously across multiple geographically dispersed clusters. Cached data is federated across clusters to provide redundancy, off-site backup, and multiple points of access for application users in different geographical locations. For complete details about Coherence Federation, see Federating Caches Across Clusters in *Administering Oracle Coherence*. For additional information, see Configure Coherence Federation in the *Oracle WebLogic Remote Console Online Help*.

Use the **Coherence Federation** tab on the **Coherence Cluster** settings page to enable a federation topology and to configure a remote cluster participant to which caches are federated. When selecting a topology, a topology configuration is automatically created and named `Default-Topology`. Federation must be configured on both the local cluster participant and the remote cluster participant. At least one host on the remote cluster must be provided. If

a custom port is being used on the remote cluster participant, then change the cluster port accordingly. Managed Coherence servers must be restarted for federation changes to take affect. The `CoherenceFederationParamsBean` MBean also exposes the cluster federation parameters and can be used to configure cache federation.

> **Note**
>
> - The `Default-Topology` topology configuration is created and used if no federation topology is specified in the cache configuration file.
> - When using federation, matching topologies must be configured on both the local and remote clusters. For example, selecting `none` for the topology in a local cluster and `active-active` as the topology in the remote cluster can lead to unpredictable behavior. Similarly, if a local cluster is set to use active-passive, then the remote cluster must be set to use passive-active.

Configuring Managed Coherence Servers

Managed Coherence servers expose several cluster member settings that can be configured for a specific domain.

Many of the settings use default values that can be changed as required. The instructions in this section assume that a managed server has already been created and associated with a Coherence cluster. For details on creating managed Coherence servers, see Create Standalone Managed Coherence Servers.

Use the **Coherence** tab on a managed server's setting page to configure Coherence cluster member settings. A `CoherenceMemberConfig` MBean is created for each managed server and exposes the Coherence cluster member parameters.

> **Note**
>
> WLS configuration take precedence over Coherence system properties. In general, Coherence configuration in WLS should be changed using WLST or a Coherence cluster configuration file instead of using the system properties.

Use the following tasks to configure a managed Coherence server:

Configure Coherence Cluster Member Storage Settings

The storage settings for managed Coherence servers can be configured as required. Enabling storage on a server means the server is responsible for storing a portion of both primary and backup data for the Coherence cluster. Servers that are intended to store data must be configured as storage-enabled servers. Servers that host cache applications and cluster proxy servers should be configured as storage-disabled servers and are typically not responsible for storing data because sharing resource can become problematic and affect application and cluster performance.

> **Note**
>
> If a managed Coherence server is part of a WebLogic Server cluster, then the Coherence storage settings that are specified on the WebLogic Server cluster override the storage settings on the server. The storage setting is an exception to the general rule that server settings override WebLogic Server cluster settings. Moreover, the final runtime configuration is not reflected in the console. Therefore, a managed Coherence server may show that storage is disabled even though storage has been enabled through the Coherence tab for a WebLogic Server cluster. Always check the WebLogic Server cluster settings to determine whether storage has been enabled for a managed Coherence server.

Use the following fields on the **Advanced**: **Coherence** tab to configure storage settings:

- **Local Storage Enabled** – This field specifies whether a managed Coherence server to stores data. If this option is not selected, then the managed Coherence server does not store data and is considered a cluster client.
- **Coherence Web Local Storage Enabled** – This field specifies whether a managed Coherence server stores HTTP session data. For details on using Coherence to store session data, see Using Coherence*Web with WebLogic Server in *Administering HTTP Session Management with Oracle Coherence*Web*.

Configure Coherence Cluster Member Unicast Settings

Managed Coherence servers communicate with each other using unicast (point-to-point) communication. Unicast is used even if the cluster is configured to use multicast communication. For details on unicast in Coherence, see Specifying a Cluster Member's Unicast Address in *Developing Applications with Oracle Coherence*.

Use the following fields on the **Advanced**: **Coherence** tab to configure unicast settings:

- **Unicast Listen Address** – This field specifies the address on which the server listens for unicast communication. If no address is provided, then a routable IP address is automatically selected. The address field also supports Classless Inter-Domain Routing (CIDR) notation, which uses a subnet and mask pattern for a local IP address to bind to instead of specifying an exact IP address.
- **Unicast Listen Port** – This field specifies the ports on which the server listens for unicast communication. A cluster member uses two unicast UDP ports which are automatically assigned from the operating system's available ephemeral port range (as indicated by a value of `0`). The default value ensures that Coherence cannot accidentally cause port conflicts with other applications. However, if a firewall is required between cluster members (an atypical configuration), then a port can be manually assigned and a second port is automatically selected (*port1* +1).
- **Unicast Port Auto Adjust** – This field specifies whether the port automatically increments if the port is already in use.

Use Dynamic Management

A Coherence cluster can be managed from any JMX-compatible client such as JConsole or Java VisualVM. The management information includes runtime statistics and operational settings. The management information is specific to the Coherence management domain and is different than the management information that is provided for Coherence as part of the

WebLogic management domain. For a detailed reference of Coherence MBeans, see Oracle Coherence MBeans Reference in *Managing Oracle Coherence*.

Coherence is configured to start in dynamic management mode. One cluster member is automatically selected as a management proxy and is responsible for aggregating the management information from all other cluster members. The Administration server for the WebLogic domain then integrates the management information and it is made available through the domain runtime MBean server. If the cluster member is not operational, then another cluster member is automatically selected as the management proxy.

At runtime, use a JMX client to connect to the domain runtime MBean server where the Coherence management information is located within the Coherence management namespace. For details about connecting to the domain runtime MBean server, see Accessing WebLogic Server MBeans with JMX in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.

Configure Coherence Cluster Member Identity Settings

A set of identifiers are used to give a managed Coherence server an identity within the cluster. The identity information is used to differentiate servers and conveys the server's role within the cluster. Some identifiers are also used by the cluster service when performing cluster tasks. Lastly, the identity information is valuable when displaying management information (for example, JMX) and facilitates interpreting log entries.

Use the following fields on the **Advanced**: **Coherence** tab to configure member identity settings:

- **Site Name** – This field specifies the name of the geographic site that hosts the managed Coherence server. The server's domain name is used if no name is specified. For WAN clustering, this value identifies the data center where the member is located. The site name can be used as the basis for intelligent routing, load balancing, and disaster recovery planning (that is, the explicit backing up of data on separate geographic sites). The site name also helps determine where to back up data when using distributed caching and the default partition assignment strategy. Lastly, the name is useful for displaying management information (for example, JMX) and interpreting log entries.
- **Rack Name** – This field specifies the name of the location within a geographic site that the managed Coherence server is hosted at and is often a cage, rack, or bladeframe identifier. The rack name can be used as the basis for intelligent routing, load balancing, and disaster recovery planning (that is, the explicit backing up of data on separate bladeframes). The rack name also helps determine where to back up data when using distributed caching and the default partition assignment strategy. Lastly, the name is useful for displaying management information (for example, JMX) and interpreting log entries.
- **Role Name** – This field specifies the managed Coherence server's role in the cluster. The role name allows an application to organize cluster members into specialized roles, such as storage-enabled or storage-disabled.

  If a managed Coherence server is part of a WebLogic Server cluster, the cluster name is automatically used as the role name and this field cannot be set. If no name is provided, the default role name that is used is `WebLogicServer`.

Configure Coherence Cluster Member Logging Levels

Logging levels can be configured for each managed Coherence server. The default log level is D5 and can be changed using the server's Logging tab. For details on WebLogic Server logging, see *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*.

To configure a managed Coherence server's logging level:

1. From the Summary of Servers screen, select a managed Coherence server.
2. On the server's settings page, select the **Logging** tab, then select the **Platform Logger Levels** subtab.
3. On the **Platform Logger Levels** page, click the **+** (plus) sign to add a logging level.

Value	Resultant Message Displays
`com.oracle.coherence=FINEST`	D9
`com.oracle.coherence=INFO`	D3
No Value	D5 (Default)

4. Click **Save**.

Using a Single-Server Cluster

A single-server cluster is a cluster that is constrained to run on a single managed server instance and does not access the network.

The server instance acts as a storage-enabled cluster member, a client, and a proxy. A single-server cluster is easy to set up and offers a quick way to start and stop a cluster. A single-server cluster is used during development and should not be used for production or testing environments.

To create a single-server cluster:

- Create a Coherence cluster and select a managed server instance to be a member of the cluster. The administration server instance can be used to facilitate setup. See Define a Coherence Cluster Resource.
- Configure the cluster and set the Time To Live value to `0` if using multicast communication. See Configure Cluster Communication.
- Configure the managed server instance and set the unicast address to an address that is routed to loop back. On most computers, setting the address to `127.0.0.1` works. See Configure Coherence Cluster Member Unicast Settings.

For detailed information, see Configure and Deploy Coherence on a Single-Server Cluster in the *Oracle WebLogic Remote Console Online Help*.

Using WLST with Coherence

The WLST is a command-line interface that you can use to automate domain configuration tasks, including configuring and managing Coherence clusters.
For more information, see Understanding the WebLogic Scripting Tool.

Setting Up Coherence with WLST (Offline)

WLST can be used to set up Coherence clusters. The following examples demonstrate using WLST in offline mode to create and configure a Coherence cluster. It is assumed that a domain has already been created and that the examples are completed in the order in which they are presented. In addition, the examples only create a data tier. Additional tiers can be created as required. Lastly, the examples are not intended to demonstrate every Coherence MBean. For a complete list of Coherence MBeans, see *MBean Reference for Oracle WebLogic Server*.

```
readDomain('/ORACLE_HOME/user_projects/domains/base_domain')
```

Create a Coherence Cluster

```
create('myCoherenceCluster', 'CoherenceClusterSystemResource')
```

Create a Tier of Managed Coherence Servers

```
create('coh_server1', 'Server')
cd('Server/coh_server1')
set('ListenPort', 7005)
set('ListenAddress', '192.168.0.100')
set('CoherenceClusterSystemResource', 'myCoherenceCluster')

cd('/')
create('coh_server2','Server')
cd('Server/coh_server2')
set('ListenPort', 7010)
set('ListenAddress', '192.168.0.101')
set('CoherenceClusterSystemResource', 'myCoherenceCluster')

cd('/')
create('DataTier', 'Cluster')
assign('Server', 'coh_server1,coh_server2','Cluster','DataTier')
cd('Cluster/DataTier')
set('MulticastAddress', '237.0.0.101')
set('MulticastPort', 8050)

cd('/CoherenceClusterSystemResource/myCoherenceCluster')
set('Target', 'DataTier')
```

Configure Coherence Cluster Parameters

```
cd('CoherenceClusterSystemResource/myCoherenceCluster/CoherenceResource/
myCoherenceCluster/CoherenceClusterParams/NO_NAME_0')
set('ClusteringMode', 'unicast')
set('SecurityFrameworkEnabled','false')
set('ClusterListenPort', 7574)
```

Configure Well Known Addresses

```
create('wka_config','CoherenceClusterWellKnownAddresses')
cd('CoherenceClusterWellKnownAddresses/NO_NAME_0')

create('WKA1','CoherenceClusterWellKnownAddress')
cd('CoherenceClusterWellKnownAddress/WKA1')
set('ListenAddress', '192.168.0.100')
cd('../..')

create('WKA2','CoherenceClusterWellKnownAddress')
cd('CoherenceClusterWellKnownAddress/WKA2')
set('ListenAddress', '192.168.0.101')
```

Set Logging Properties

```
cd('/')
cd('CoherenceClusterSystemResource/myCoherenceCluster/CoherenceResource/
myCoherenceCluster')
create('log_config)','CoherenceLoggingParams')
cd('CoherenceLoggingParams/NO_NAME_0')
set('Enabled', 'true')
set('LoggerName', 'com.oracle.coherence')
```

Configure Managed Coherence Servers

```
cd('/')
cd('Servers/coh_server1')
create('member_config', 'CoherenceMemberConfig')
cd('CoherenceMemberConfig/member_config')
set('LocalStorageEnabled', 'true')
set('RackName', '100A')
set('RoleName', 'Server')
set('SiteName', 'pa-1')
set('UnicastListenAddress', '192.168.0.100')
set('UnicastListenPort', 0)
set('UnicastPortAutoAdjust', 'true')

cd('/')
cd('Servers/coh_server2')
create('member_config', 'CoherenceMemberConfig')
cd('CoherenceMemberConfig/member_config')
set('LocalStorageEnabled', 'true')
set('RackName', '100A')
set('RoleName', 'Server')
set('SiteName', 'pa-1')
set('UnicastListenAddress', '192.168.0.101')
set('UnicastListenPort', 0)
set('UnicastPortAutoAdjust', 'true')

updateDomain()
closeDomain()
```

Setting the Cluster Name and Port

```
readDomain('/ORACLE_HOME/user_projects/domains/base_domain')

cd('CoherenceClusterSystemResource/myCoherenceCluster/CoherenceResource/
myCoherenceCluster)
set('Name', 'MyCluster')

cd('CoherenceClusterSystemResource/myCoherenceCluster/CoherenceResource/
myCoherenceCluster/CoherenceClusterParams/NO_NAME_0')
set('ClusterListenPort', 9123)

updateDomain()
closeDomain()
```

Accessing Coherence MBeans by Using WLST

When running Coherence within WebLogic Server in a managed Coherence Servers environment, WebLogic Server domain runtime MBean server collects JMX information from the management proxy and this information is accessible by using WLST.

After you have connected using WLST and switched to domainRuntime(), the MBeanServer is available through the `mbs` object, where you can perform queries and operations on standard Coherence MBeans.

Given below are samples on how to access various MBean values and operations. Note that this is not an exhaustive list and is provided to illustrate how to access MBeans by using WLST. For a list of Coherence MBeans, including their attributes and operations, see Oracle Coherence MBeans Reference in *Managing Oracle Coherence*.

> **Note**
>
> As WebLogic Server adds an additional **location** key when returning MBeans, some of the examples use the following WLST function to return the fully qualified name, including this **location** key.

```
# Return the fully qualified Name for a query
def coh_getFullyQualifiedName(query):
   beans = list(mbs.queryMBeans(ObjectName(query),None))
   if len(beans) == 0:
      raise RuntimeException('No results found')
   for bean in beans:
      return bean.getObjectName()
```

Example 11-1 Access a single MBean such as the `ClusterMBean`

This example shows how to access `ClusterMBean` and display various attributes.

```
# Return the fully qualified Name for a query
def coh_getFullyQualifiedName(query):
   beans = list(mbs.queryMBeans(ObjectName(query),None))
   if len(beans) == 0:
      raise RuntimeException('No results found')
   for bean in beans:
      return bean.getObjectName()

#
## Entry point
#
connect ("username","password", "t3://host:port")
# Enter the WebLogic Server administrator username, password, and the
administration server host and port.

domainRuntime()

query = coh_getFullyQualifiedName('Coherence:type=Cluster,*')
print 'Fully qualified query: ' + str(query)
beans = list(mbs.queryMBeans(query, None))

# Should only be one cluster MBean
if len(beans) == 0:
   print 'Unable to find ClusterMBean'
else:
   # Normally there is only one ClusterMBean but if you had multiple
   # Coherence clusters then > 1 could be returned
   bean = beans[0]
   cluster = bean.getObjectName()
   clusterName = mbs.getAttribute(cluster, 'ClusterName')
   clusterSize = mbs.getAttribute(cluster, 'ClusterSize')
   clusterVersion = mbs.getAttribute(cluster, 'Version')

   print 'Cluster Name:      ' + clusterName
```

```
   print 'Cluster Size:      ' + str(clusterSize)
   print 'Coherence Version: ' + clusterVersion
```

Sample Output:

```
Location changed to domainRuntime tree. This is a read-only tree
with DomainMBean as the root MBean.
For more help, use help('domainRuntime')

Fully qualified query:
Coherence:cluster=CoherenceCluster,Location=storage1,type=Cluster
Cluster Name:      CoherenceCluster
Cluster Size:      4
```

Example 11-2 Display Information from Multiple MBeans such as the `CacheMBean`

This example shows how to access the CoherenceMBeans for all defined caches.

```
connect ("username","password", "t3://host:port")
# Enter the WebLogic Server administrator username, password, and the
administration server host and port.

domainRuntime()

beans = list(mbs.queryMBeans(ObjectName('Coherence:type=Cache,*'),None))
for bean in beans:
   cache = bean.getObjectName()
   cacheSize = mbs.getAttribute(cache, 'Size')
   cacheUnits = mbs.getAttribute(cache, 'Units')

   print 'Mbean: ' + str(cache)
   print 'Size:  ' + str(cacheSize)
   print 'Units: ' + str(cacheUnits)
```

Sample Output

```
Mbean:
Coherence:Location=storage1,nodeId=1,type=Cache,cluster=CoherenceCluster,servi
ce="ExampleGAR:PartitionedPofCache",member=storage1,name=control,tier=back
Size:  0
Units: 0
Mbean:
Coherence:Location=storage1,nodeId=2,type=Cache,cluster=CoherenceCluster,servi
ce="ExampleGAR:PartitionedPofCache",member=storage2,name=stores,tier=back
Size:  12
Units: 3488
Mbean:
Coherence:Location=storage1,nodeId=1,type=Cache,cluster=CoherenceCluster,servi
ce="ExampleGAR:PartitionedPofCache",member=storage1,name=contacts,tier=back
Size:  10
Units: 3360
Mbean:
Coherence:Location=storage1,nodeId=2,type=Cache,cluster=CoherenceCluster,servi
ce="ExampleGAR:PartitionedPofCache",member=storage2,name=contacts,tier=back
```

```
Size:  10
Units: 3352
Mbean:
Coherence:Location=storage1,nodeId=2,type=Cache,cluster=CoherenceCluster,servi
ce="ExampleGAR:PartitionedPofCache",member=storage2,name=control,tier=back
Size:  0
Units: 0
Mbean:
Coherence:Location=storage1,nodeId=1,type=Cache,cluster=CoherenceCluster,servi
ce="ExampleGAR:PartitionedPofCache",member=storage1,name=stores,tier=back
Size:  8
Units: 2328
```

Example 11-3 Invoke an Operation against an MBean to force Persistence Recovery

This example shows how to force Persistence Recovery on a specific service by invoking the `forceRecovery` operation.

```
# Return the Persistence coordinator MBean
def coh_getPersistenceCoordinator(serviceName):
   query = 'Coherence:type=Persistence,service=' + serviceName +
',responsibility=PersistenceCoordinator,*'
   beans = list(mbs.queryMBeans(ObjectName(query), None))
   if len(beans) == 0:
      raise RuntimeException('No results found')
   for bean in beans:
      return bean.getObjectName()

##
## Entry Point
##

connect("username","password", "t3://host:port")
# Enter the WebLogic Server administrator username, password, and the
administration server host and port.

domainRuntime()

# if serviceName includes ':' it must be quoted
serviceName = '"ExampleGAR:PartitionedPofCache"'
objectName = coh_getPersistenceCoordinator(serviceName)
print 'Coordinator is ' + str(objectName)

mbs.invoke(objectName, 'forceRecovery', None, None)

print 'Force Recovery invoked'
```

Example 11-4 Invoke an Operation which returns a value

This example shows how to invoke an MBean operation, which returns a value by invoking `reportScheduledDistributions` on the `PartitionAssignment` MBean.

```
# Return the fully qualified Name for a query
def coh_getFullyQualifiedName(query):
   beans = list(mbs.queryMBeans(ObjectName(query),None))
```

```
   if len(beans) == 0:
      raise RuntimeException('No results found')
   for bean in beans:
      return bean.getObjectName()

##
## Entry Point
##

connect("username","password", "t3://host:port")
# Enter the WebLogic Server administrator username, password, and the
administration server host and port.

domainRuntime()

# if serviceName includes ':' it must be quoted
serviceName = '"ExampleGAR:PartitionedPofCache"'
partitionAssignment =
coh_getFullyQualifiedName('Coherence:type=PartitionAssignment,service=' +
serviceName + ',responsibility=DistributionCoordinator,*')
print 'Partition Assignment is ' + str(partitionAssignment)

print mbs.invoke(partitionAssignment, 'reportScheduledDistributions',
[java.lang.Boolean("true")], ["boolean"])
```

Sample Output

```
Location changed to domainRuntime tree. This is a read-only tree
with DomainMBean as the root MBean.
For more help, use help('domainRuntime')

Partition Assignment is
Coherence:service="ExampleGAR:PartitionedPofCache",cluster=CoherenceCluster,re
sponsibility=DistributionCoordinator,Location=storage1,type=PartitionAssignmen
t
No distributions are currently scheduled for this service.
```

Example 11-5 Invoke a Federation Operation

This example shows how to invoke a `Federation` operation for a service.

```
# Return the fully qualified Name for a query
def coh_getFullyQualifiedName(query):
   beans = list(mbs.queryMBeans(ObjectName(query),None))
   if len(beans) == 0:
      raise RuntimeException('No results found')
   for bean in beans:
      return bean.getObjectName()

##
## Entry Point
##

connect("username","password", "t3://host:port")
# Enter the WebLogic Server administrator username, password, and the
```

```
administration server host and port.

domainRuntime()

# if serviceName includes ':' it must be quoted
serviceName   = '"ExampleGAR:PartitionedPofCache"'
targetCluster = 'Boston'
command       = 'start'

# Other Federation commands could be stop, pause or replicateAll

federation = coh_getFullyQualifiedName('Coherence:type=Federation,service=' +
serviceName + ',responsibility=Coordinator,*')
print 'Federation MBean is ' + str(federation)

print mbs.invoke(federation, command, [java.lang.String(target)],
["java.lang.String"])
```

Sample Output

```
Location changed to domainRuntime tree. This is a read-only tree
with DomainMBean as the root MBean.
For more help, use help('domainRuntime')

Federation Mbean is
Coherence:service="ExampleGAR:PartitionedPofCache",cluster=CoherenceCluster,re
sponsibility=Coordinator,Location=storage1,type=Federation
```

Persisting Coherence Caches with WLST

WLST includes a set of commands that can be used to persist and recover cached data from disk. The commands are automatically available when connected to an Administration Server domain runtime MBean server.

For more information about Coherence cache persistence, see Persisting Caches in *Administering Oracle Coherence*.

Table 11-2 lists WLST commands for persisting Coherence caches. Example 11-6 demonstrates using the commands.

Table 11-2 WLST Coherence Persistence Commands

Command	Description
`coh_createSnapshot(`*`snapshotName`*`, `*`serviceName`*`)`	Persist the data partitions of a service to disk • *snapshotName* – any user defined name • *serviceName* – the name of the partitioned or federated cache service for which the snapshot is created

Table 11-2 (Cont.) WLST Coherence Persistence Commands

Command	Description
`coh_recoverSnapshot(`*`snapshotName, serviceName`*`)`	Restore the data partitions of a service from disk. Any existing data in the caches of a service are lost. • *snapshotName* – the name of a snapshot to recover • *serviceName* – the name of the partitioned or federated cache service for which the snapshot was created
`coh_listSnapshots(`*`serviceName`*`)`	Return a list of available snapshots • *serviceName* – the name of the partitioned or federated cache service for which the snapshots are listed
`coh_validateSnapshot(`*`snapshotDir, verbose`*`)`	Check whether a snapshot is complete and without error • *snapshotDir* – The full path to a snapshot including the snapshot name. The default snapshot location is *USER_HOME*`/coherence/snapshot`. • *verbose* – return more detailed validation information
`coh_archiveSnapshot(`*`snapshotName, serviceName`*`)`	Save a snapshot to a central location. The location is specified in the snapshot archiver definition that is associated with a service. • *snapshotName* – the name of a snapshot to archive • *serviceName* – the name of the partitioned or federated cache service for which the snapshot was created
`coh_retrieveArchivedSnapshot(`*`snapshotName, serviceName`*`)`	Retrieve an archived snapshot so that it can be recovered using the coh_recoverSnapshot command • *snapshotName* – the name of a snapshot to retrieve • *serviceName* – the name of the partitioned or federated cache service for which the snapshot was archived
`coh_listArchivedSnapshots(`*`serviceName`*`)`	Return a list of available archived snapshots • *serviceName* – the name of the partitioned or federated cache service for which the snapshot was archived

Table 11-2 (Cont.) WLST Coherence Persistence Commands

Command	Description
`coh_validateArchivedSnapshot(snapshotName, clusterName, serviceName, achviverName, verbose)`	Check whether an archived snapshot is complete and without error. The operational override configuration file containing the archiver must be available on the classpath. • *snapshotName* – the name of an archived snapshot to validate • *clusterName* – the name of the cluster where the partitioned or federated cache service is running • *serviceName* – the name of the partitioned or federated cache service for which the archived snapshot was created • *archiverName* – the name of the snapshot archiver definition that is being used by the service. • *verbose* – return more detailed validation information
`coh_removeArchivedSnapshot(snapshotName, serviceName)`	Delete an archived snapshot from disk • *snapshotName* – the name of an archived snapshot to delete • *serviceName* – the name of the partitioned or federated cache service for which the archived snapshot is deleted
`coh_removeSnapshot(snapshotName, serviceName)`	Delete a snapshot from disk • *snapshotName* – the name of a snapshot to delete • *serviceName* – the name of the partitioned or federated cache service for which the snapshot is deleted

Note

If the *serviceName* includes a colon (`:`), then enclose it in double quotation marks. For example:

```
coh_createSnapshot('Snapshot1', '"ExampleGAR:PartitionedPofCache"')
```

Example 11-6 demonstrates using the persistence API from WLST to persist the caches for a partitioned cache service.

Example 11-6 WLST Example for Persisting Caches

```
serviceName   = '"ExampleGAR:ExamplesPartitionedPofCache"';
snapshotName  = 'new-snapshot'

connect('weblogic','password','t3://machine:7001')

# Must be in domain runtime tree otherwise no MBeans are returned
```

```
domainRuntime()

try:
   coh_listSnapshots(serviceName)
   coh_createSnapshot(snapshotName, serviceName)
   coh_listSnapshots(serviceName)
   coh_recoverSnapshot(snapshotName, serviceName)
   coh_archiveSnapshot(snapshotName, serviceName)
   coh_listArchivedSnapshots(serviceName)
   coh_removeSnapshot(snapshotName, serviceName)
   coh_retrieveArchivedSnapshot(snapshotName, serviceName)
   coh_recoverSnapshot(snapshotName, serviceName)
   coh_listSnapshots(serviceName)
except PersistenceException, rce:
   print 'PersistenceException: ' + str(rce)
except Exception,e:
   print 'Unknown Exception' + str(e)
else:
   print 'All operations complete'
```

12

Clustering Best Practices

Learn about the recommended design and deployment practices that maximize the scalability, reliability, and performance of applications hosted by an Oracle WebLogic Server cluster.

This chapter includes the following sections:

General Design Considerations

Learn general design guidelines for clustered applications.

Strive for Simplicity

Distributed systems are complicated by nature. For a variety of reasons, make simplicity a primary design goal. Minimize "moving parts" and do not distribute algorithms across multiple objects.

Minimize Remote Calls

You can improve the performance and reduce the effects of failures by minimizing remote calls.

Transfer Objects Reduce Remote Calls

EJBs consume significant system resources and network bandwidth to execute—they are unlikely to be the appropriate implementation for every object in an application.

Use EJBs to model logical groupings of an information and associated business logic. For example, use an EJB to model a logical subset of the line items on an invoice—for instance, items to which discounts, rebates, taxes, or other adjustments apply.

In contrast, an individual line item in an invoice is fine-grained—implementing it as an EJB wastes network resources. Implement objects that simply represents a set of data fields, which require only `get` and `set` functionality, as *transfer objects.*

Transfer objects (sometimes referred to as *value objects* or *helper classes*) are good for modeling entities that contain a group of attributes that are always accessed together. A transfer object is a serializable class within an EJB that groups related attributes, forming a composite value. This class is used as the return type of a remote business method.

Clients receive instances of this class by calling coarse-grained business methods, and then locally access the fine-grained values within the transfer object. Fetching multiple values in one server round-trip decreases network traffic and minimizes latency and server resource usage.

Distributed Transactions Increase Remote Calls

Avoid transactions that span multiple server instances. Distributed transactions issue remote calls and consume network bandwidth and overhead for resource coordination.

Web Application Design Considerations

Learn design considerations for clustered servlets and JSPs.

Configure In-Memory Replication

To enable automatic failover of servlets and JSPs, session state must persist in memory. For instructions to configure in-memory replication for HTTP session states, see Requirements for HTTP Session State Replication, and Configure In-Memory HTTP Replication.

Design for Idempotence

Failures or impatient users can result in duplicate servlet requests. Design servlets to tolerate duplicate requests.

Programming Considerations

For more information, see Programming Considerations for Clustered Servlets and JSPs.

EJB Design Considerations

Learn design considerations for clustered RMI objects.

Design Idempotent Methods

It is not always possible to determine when a server instance failed with respect to the work it was doing at the time of failure. For instance, if a server instance fails after handling a client request but before returning the response, there is no way to tell that the request was handled. A user that does not get a response retries, resulting in an additional request.

Failover for RMI objects requires that methods be *idempotent*. An idempotent method is one that can be repeated with no negative side-effects.

Follow Usage and Configuration Guidelines

Table 12-1 summarizes usage and configuration guidelines for EJBs.

Table 12-1 EJB Types and Guidelines

Object Type	Usage	Configuration
EJBs of all types	Use EJBs to model logical groupings of an information and associated business logic. See Transfer Objects Reduce Remote Calls.	For configuration, see clusterable homes in Table 12-2.

Table 12-1 (Cont.) EJB Types and Guidelines

Object Type	Usage	Configuration
Stateful Session Beans	Recommended for high volume, heavy-write transactions. Remove stateful session beans when finished to minimize EJB container overhead. A stateful session bean instance is associated with a particular client and remains in the container until explicitly removed by the client or by the container when it times out. Meanwhile, the container may passivate inactive instances to disk. This consumes overhead and can affect performance. **Note:** Although unlikely, the current state of a stateful session bean can be lost. For example, if a client commits a transaction involving the bean and there is a failure of the primary server before the state change is replicated, the client will failover to the previously-stored state of the bean.	For configuration, see clusterable homes in Table 12-2. For configuration, see in-memory replication for EJBs in Table 12-2.
Stateless Session Beans	Scale better than stateful session beans which are instantiated on a per client basis, and can multiply and consume resources rapidly. When a home creates a stateless session bean, it returns a replica-aware stub that can route to any server where the bean is deployed. Because a stateless session bean holds no state on behalf of the client, the stub is free to route any call to any server that hosts the bean.	For configuration, see clusterable homes in Table 12-2. For configuration, see Cluster Address in Table 12-2. Configure methods to be idempotence (see Table 12-2) to support failover during method calls. (Failover is default behavior if failure occurs between method calls or if the method fails to connect to a server). The methods on stateless session bean homes are automatically set to be idempotent. It is not necessary to explicitly specify them as idempotent.
Read-only Entity Beans	Recommended whenever stale data is tolerable; suitable for product catalogs and most of the content within many applications. Reads are performed against a local cache that is invalidated on a timer basis. Read-only entities perform three to four times faster than transactional entities. **Note:** A client can successfully call setter methods on a read-only entity bean, however the data will never be moved into the persistent store.	For configuration, see clusterable homes in Table 12-2. For configuration, see Cluster Address in Table 12-2. Methods are configured to be idempotent by default.
Read-Write Entity Beans	Best suited for shared persistent data that is not subject to heavy request and update.If the access/update load is high, consider session beans and JDBC. Recommended for applications that require high data consistency, for instance, customer account maintenance. All reads and writes are performed against the database. Use the `isModified` method to reduce writes. For read-mostly applications, characterized by frequent reads and occasional updates (for instance, a catalog), a combination of read-only and read-write entity beans that extend the read-only entity beans is suitable. The read-only entity bean provides fast, weakly consistent reads, while the read-write entity bean provides strongly consistent writes.	For configuration, see clusterable homes in Table 12-2. Configure methods to be idempotence (see Table 12-2) to support failover during method calls. (Failover is default behavior if failure occurs between method calls or if the method fails to connect to a server). The methods on read-only entity beans are automatically set to be idempotent.

Cluster-Related Configuration Options

Table 12-2 lists key behaviors that you can configure for a cluster, and the associated method of configuration.

Table 12-2 Cluster-Related Configuration Options

Configurable Behavior or Resource	How to Configure
clusterable homes	Set `home-is-clusterable` in `weblogic-ejb-jar.xml` to "true".
idempotence	At bean level, set `stateless-bean-methods-are-idempotent` in `weblogic-ejb-jar.xml` to "true". At method level, set `idempotent-methods` in `weblogic-ejb-jar.xml`
in-memory replication for EJBs	Set `replication-type` in `weblogic-ejb-jar.xml` to "InMemory".
Cluster Address	The cluster address identifies the Managed Servers in the cluster. It is used in stateless beans to construct the host name portion of URLs. It can be assigned explicitly, or generated automatically by WebLogic Server for each request. See Cluster Address.
`clients-on-same-server`	Set `clients-on-same-server` in `weblogic-ejb-jar.xml` to "true" if all clients accessing the EJB will do so from the same server on which the bean is deployed. If `clients-on-same-server` is "True" the server instance will not multicast JNDI announcements for the EJB when it is deployed, hence reducing the startup time for a large clusters.
Custom load balancing for stateful session EJBs and stateless session	Use `home-call-router-class-name` in `weblogic-ejb-jar.xml` to specify the name of a custom class to use for routing bean method calls for these types of beans. This class must implement `weblogic.rmi.cluster.CallRouter()`. See The WebLogic Cluster API .
Custom load balancing for stateless session bean	Use `stateless-bean-call-router-class-name` in `weblogic-ejb-jar.xml` to specify the name of a custom class to use for routing stateless session bean method calls. This class must implement `weblogic.rmi.cluster.CallRouter()`. See The WebLogic Cluster API .
Configure stateless session bean as clusterable	Set `stateless-bean-is-clusterable` in `weblogic-ejb-jar.xml` to "true" to allow the EJB to be deployed to a cluster.
Load balancing algorithm for stateless session beans	Use `stateless-bean-load-algorithm` in `weblogic-ejb-jar.xml` to specify the algorithm to use for load balancing between replicas of the EJB home. If this property is not defined, WebLogic Server uses the algorithm specified by the `weblogic.cluster.defaultLoadAlgorithm` attribute in `config.xml`.
Machine	The WebLogic Server Machine resource associates server instances with the computer on which it runs. See Configure Machine Names.
Replication groups	Replication groups allow you to control where HTTP session states are replicated. See Configure Replication Groups.

State Management in a Cluster

Different services in a WebLogic Server cluster provide varying types and degrees of state management. The four categories of services that are distinguished by how they maintain state in memory or persistent storage are as follows:

- Stateless services: A stateless service does not maintain state in memory between invocations.
- Conversational services: A conversational service is dedicated to a particular client for the duration of a session. During the session, it serves all requests from the client, and only requests from that client. Throughout a session there is generally state information that the application server must maintain between requests. Conversational services typically maintain transient state in memory, which can be lost in the event of failure. If session state is written to a shared persistent store between invocations, the service is stateless. If persistent storage of state is not required, alternatives for improving performance and scalability include:
  - Session state can be sent back and forth between the client and server under the covers, again resulting in a stateless service. This approach is not always feasible or desirable, particularly with large amounts of data.
  - More commonly, session state may be retained in memory on the application server between requests. Session state can be paged out from memory as necessary to free up memory. Performance and scalability are still improved in this case because updates are not individually written to disk and the data is not expected to survive server failures.
- Cached services: A cached service maintains state in memory and uses it to process requests from multiple clients. Implementations of cached services vary in the extent to which they keep the copies of cached data consistent with each other and with associated data in the backing store.
- Singleton services: A singleton service is active on exactly one server in the cluster at a time and processes requests from multiple clients. A singleton service is generally backed by private, persistent data, which it caches in memory. It may also maintain transient state in memory, which is either regenerated or lost in the event of failure. Upon failure, a singleton service must be restarted on the same server or migrated to a new server.

Table 12-3 summarizes how Jakarta EE and WebLogic support each of these categories of service.

Support for stateless and conversational services is described for two types of clients:

- *Loosely coupled* clients include browsers or Web Service clients that communicate with the application server using standard protocols.
- *Tightly coupled* clients are objects that run in the application tier or in the client-side environment, and communicate with the application server using proprietary protocol.

Table 12-3 Jakarta EE and WebLogic Support for Service Types

Service	Jakarta EE Support	WebLogic Server Scalability and Reliability Features for...
Stateless Service with loosely coupled clients	All Jakarta EE APIs are either stateless or may be implemented in a stateless manner by writing state information to a shared persistent store between invocations. Jakarta EE does not specify a standard for load balancing and failover. For loosely coupled clients, load balancing must be performed by external IP-based mechanisms	WebLogic Server increases the availability of stateless services by deploying multiple instances of the service to a cluster. For loosely coupled clients of a stateless service, WebLogic Server supports external load balancing solutions, and provides proxy plug-ins for session failover and load balancing. See: • Stateless Session Beans • Load Balancing HTTP Sessions with an External Load Balancer • Load Balancing with a Proxy Plug-in
Stateless Service with tightly coupled clients	These Jakarta EE APIs support tightly coupled access to the following stateless services: • JNDI (after initial access) • Factories, such as EJB homes, JDBC connection pools, and JMS connection factories • Stateless session beans	WebLogic Server increases the availability of stateless services by deploying multiple instances of the service to a cluster. For tightly coupled clients of a stateless service, WebLogic Server supports load balancing and failover in its RMI implementation. The WebLogic Server replica-aware stub for a clustered RMI object lists the server instances in the cluster that currently offer the service, and the configured load balancing algorithm for the object. WebLogic Server uses the stub to make load balancing and failover decisions. See: • Stateless Session Beans • Load Balancing for EJBs and RMI Objects
Conversational services with loosely coupled clients	These Jakarta EE APIs support loosely coupled access to the following conversational services: • Servlets • Web Services Jakarta EE does not specify a standard for load balancing and failover. Load balancing can be accomplished with external IP-based mechanisms or application server code in the presentation tier. Because protocols for conversations services are stateless, load balancing should occur only when the session is created. Subsequent requests should stick to the selected server.	WebLogic Server increases the reliability of sessions with: • Failover, based on in-memory replication of session state, and distribution of primaries and secondaries across the cluster. • Configurable replication groups, and the ability to specify preferred replication groups for hosting secondaries. • Load balancing using external load balancers or proxy-plug-ins. See • HTTP Session State Replication • Load Balancing for Servlets and JSPs

Table 12-3 (Cont.) Jakarta EE and WebLogic Support for Service Types

Service	Jakarta EE Support	WebLogic Server Scalability and Reliability Features for...
Conversational services with tightly coupled clients	The Jakarta EE standard provides EJB stateful session beans to support conversational services with tightly coupled clients.	WebLogic Server increases the availability and reliability of stateful session beans with these features: • Caching. • Persistent storage of passivated bean state. • Initial load balancing occurs when an EJB home is chosen to create the bean. The replica-aware stub is hard-wired to the chosen server, providing session affinity. • When primary/secondary replication is enabled, the stub keeps track of the secondary and performs failover. • Updates are sent from the primary to the secondary only on transaction boundaries. See Stateful Session Beans.
Cached Services	Jakarta EE does not specify a standard for cached services. Entity beans with Bean-Managed-Persistence can implement custom caches.	Weblogic Server supports caching of: **Stateful session beans** For a list of WebLogic features that increase scalability and reliability of stateful session beans, see description in the previous row. **Entity beans** Weblogic Server supports the following caching features for entity beans. • Short term or cross-transaction caching • Relationship caching • Combined caching allows multiple entity beans that are part of the same Jakarta EE application to share a single runtime cache Consistency between the cache and the external data store can be increased by: • flushing the cache • refreshing cache after updates to the external data store • invalidating the cache • concurrency control **read-mostly pattern** WebLogic Server supports the "read-mostly pattern" by combining read-only and read-write EJBs. **JSPs** WebLogic Server provides custom JSP tags to support caching at fragment or page level.

Table 12-3 (Cont.) Jakarta EE and WebLogic Support for Service Types

Service	Jakarta EE Support	WebLogic Server Scalability and Reliability Features for...
Singleton Services	Jakarta EE APIs used to implement singleton services including: • JMS Destinations • JTA transaction managers • Cached entity beans with pessimistic concurrency control Scalability can be increased by "partitioning" the service into multiple instances, each of which handles a different slice of the backing data and its associated requests.	WebLogic Server features for increasing the availability of singleton services including: • Support for multiple thread pools for servers, to harden individual servers against failures • Health monitoring and lifecycle APIs to support detection restart of failed and ailing servers • Ability to upgrade software without interrupting services • Ability to migrate JMS servers and JTA transaction recovery services.

Application Deployment Considerations

Deploy clusterable objects to the cluster, rather than to individual Managed Servers in the cluster.

For more information, see *Deploying Applications to Oracle WebLogic Server*.

Architecture Considerations

Learn about alternative cluster architectures, load balancing options, and security options.

For more information, see Cluster Architectures.

Avoiding Problems

Learn the following considerations when planning and configuring a cluster.

Naming Considerations

For guidelines about how to name and address server instances in the cluster, see Identify Names and Addresses.

Administration Server Considerations

To start up WebLogic Server instances that participate in a cluster, each Managed Server must be able to connect to the Administration Server that manages configuration information for the domain that contains the cluster. For security purposes, the Administration Server should reside within the same DMZ as the WebLogic Server cluster.

The Administration Server maintains the configuration information for all server instances that participate in the cluster. The `config.xml` file that resides on the Administration Server contains configuration data for all clustered and non-clustered servers in the Administration Server's domain. You *should not* create a separate configuration file for each server in the cluster.

The Administration Server must be available in order for clustered WebLogic Server instances to start up. Note, however, that once a cluster is running, a failure of the Administration Server does not affect ongoing cluster operation.

The Administration Server should not participate in a cluster. Whereas it should be dedicated to the following process of administering servers:

- Maintaining configuration data
- Starting and shutting down servers
- Deploying and undeploying applications

If the Administration Server also handles client requests, there is a risk of delay in accomplishing the administration tasks.

There is no benefit in clustering an Administration Server; the administrative objects are not clusterable, and will not failover to another cluster member if the Administrative Server fails. Deploying applications on an Administration Server can reduce the stability of the server and the administrative functions it provides. If an application you deploy on the Administration Server behaves unexpectedly, it could interrupt operation of the Administration Server.

For these reasons, make sure that the Administration Server's IP address is not included in the cluster-wide DNS name.

Firewall Considerations

If your configuration includes a firewall, locate your proxy server or load-balancer in your DMZ, and the cluster, both Web and EJB containers, behind the firewall. Web containers in DMZ are not recommended. See Basic Firewall for Proxy Architectures.

If you place a firewall between the servlet cluster and object cluster in a multitier architecture, bind all servers in the object cluster to public DNS names, rather than IP addresses. Binding those servers with IP addresses can cause address translation problems and prevent the servlet cluster from accessing individual server instances.

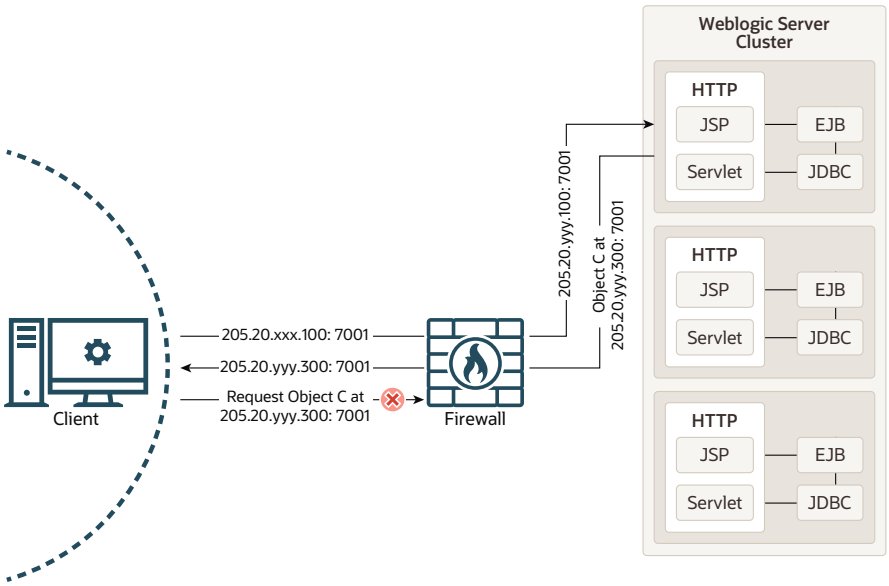
If the internal and external DNS names of a WebLogic Server instance are not identical, use the **External Listen Address** attribute for the server instance to define the server's external DNS name. Outside the firewall the **External Listen Address** should translate to external IP address of the server. Set this attribute in the WebLogic Remote Console using the **Environment**: **Servers**: *myServer* page, click **Show Advanced Fields**.

In any cluster architecture that utilizes one or more firewalls, it is critical to identify all WebLogic Server instances using publicly-available DNS names, rather than IP addresses. Using DNS names avoids problems associated with address translation policies used to mask internal IP addresses from untrusted clients.

> **Note**
>
> Use of `ExternalDNSName` is required for configurations in which a firewall is performing Network Address Translation, unless clients are accessing WebLogic Server using t3 and the default channel. For instance, `ExternalDNSName` is required for configurations in which a firewall is performing Network Address Translation, and clients are accessing WebLogic Server using HTTP via a proxy plug-in.

Figure 12-1 describes the potential problem with using IP addresses to identify WebLogic Server instances. In this figure, the firewall translates external IP requests for the subnet "xxx" to internal IP addresses having the subnet "yyy."

Figure 12-1 Translation Errors Can Occur When Servers are Identified by IP Addresses



The following steps describe the connection process and potential point of failure:

1. The client initiates contact with the WebLogic Server cluster by requesting a connection to the first server at 205.20.xxx.100:7001. The firewall translates this address and connects the client to the internal IP address of 205.20.yyy.100:7001.
2. The client performs a JNDI lookup of a pinned Object C that resides on the third WebLogic Server instance in the cluster. The stub for Object C contains the *internal* IP address of the server hosting the object, 205.20.yyy.300:7001.
3. When the client attempts to instantiate Object C, it requests a connection to the server hosting the object using IP address 205.20.yyy.300:7001. The firewall denies this connection, because the client has requested a restricted, internal IP address, rather than the publicly-available address of the server.

If there was no translation between external and internal IP addresses, the firewall would pose no problems to the client in the above scenario. However, most security policies involve hiding (and denying access to) internal IP addresses.

Evaluate Cluster Capacity Prior to Production Use

The architecture of your cluster will influence the capacity of your system. Before deploying applications for production use, evaluate performance to determine if and where you may need to add servers or server hardware to support real-world client loads. Testing software such as LoadRunner from Mercury Interactive allows you to simulate heavy client usage.

13

Troubleshooting Common Problems

Learn the guidelines to prevent Oracle WebLogic Server cluster problems or troubleshoot them if they occurs.

For more information about troubleshooting IP multicast configuration problems, see Troubleshooting Multicast Configuration.

This chapter includes the following sections:

Before You Start the Cluster

Do the following checks before you start the cluster to prevent the problems.

Check the Server Version Numbers

During steady-state operation, all servers in the cluster should be at the same maintenance level, such as:

- Same major and minor version number
- Same Patch Set number
- Same Patch Set Update number
- Same Interim/One-Off Patches

Rolling upgrade (applying maintenance to servers sequentially within a cluster) is supported in WebLogic Server for applying the following:

- Interim/One-Off Patches
- Patch Set Updates (PSUs)

The cluster's Administration Server is typically not configured as a cluster member, but it should generally run at the same maintenance level as the Managed Servers. There may be situations where the Administration Server manages multiple clusters within a single domain, which may be at different maintenance levels. In this case, the Administration Server should be at the highest maintenance level of the Managed Servers within the domain.

Check the Multicast Address

A problem with the multicast address is one of the most common reasons a cluster does not start or a server fails to join a cluster.

A multicast address is required for each cluster. The multicast address can be an IP number between 224.0.0.0 and 239.255.255.255, or a host name with an IP address within that range.

In the WebLogic Remote Console, on the **Messaging** page for the cluster, check the cluster's multicast address and port.

For each cluster on a network, the combination of multicast address and port must be unique. If two clusters on a network use the same multicast address, they should use different ports. If

the clusters use different multicast addresses, they can use the same port or accept the default port, `7001`.

Before startting the cluster, make sure the cluster's multicast address and port are correct and do not conflict with the multicast address and port of any other clusters on the network.

The errors you are most likely to see if the multicast address is bad are:

```
Unable to create a multicast socket for clustering Multicast socket send error
Multicast socket receive error
```

Check the CLASSPATH Value

Make sure the value of `CLASSPATH` is the same in all Managed Servers in the cluster. `CLASSPATH` is set by the `setEnv` script, which you run before the `startManagedWebLogic` script to start the Managed Servers.

By default, `setEnv` sets this value for `CLASSPATH` (as represented on Windows systems):

```
set WL_HOME=C:\Oracle\Middleware\Oracle_Home\wlserver
set JAVA_HOME=C:\Program Files\Java\jdk-17
.
.
set CLASSPATH=%JAVA_HOME%\lib\tools.jar;
    %WL_HOME%\server\lib\weblogic_sp.jar;
    %WL_HOME%\server\lib\weblogic.jar;
    %CLASSPATH%
```

If you change the value of `CLASSPATH` in one Managed Server, or change how `setEnv` sets `CLASSPATH`, you must change it in all Managed Servers in the cluster.

After You Start the Cluster

After you start a cluster, you can troubleshoot problems using the following methods.

Check Your Commands

If the cluster fails to start, or a server fails to join the cluster, the first step is to check any commands you have entered, such as `startManagedWebLogic` or a `java` interpreter command, for errors and misspellings.

Generate a Log File

Before contacting Oracle for help with cluster-related problems, collect diagnostic information. The most important information is a log file with multiple thread dumps from a Managed Server. The log file is especially important for diagnosing cluster freezes and deadlocks.

> **Note**
>
> *A log file that contains multiple thread dumps is a prerequisite for diagnosing your problem.*

1. Stop the server.

2. Remove or back up any log files you currently have. You should create a new log file each time you boot a server, rather than appending to an existing log file.
3. Start the server with this command, which turns on verbose garbage collection and redirects both the standard error and standard output to a log file:

   ```
   % java -ms64m -mx64m -verbose:gc -classpath $CLASSPATH
       -Dweblogic.domain=mydomain -Dweblogic.Name=clusterServer1
       -Djava.security.policy==$WL_HOME/lib/weblogic.policy
       -Dweblogic.admin.host=192.168.0.101:7001
        weblogic.Server >> logfile.txt
   ```

   Redirecting both standard error and standard output places thread dump information in the proper context with server informational and error messages and provides a more useful log.
4. Continue to run the cluster until you reproduce the problem.
5. If a server hangs, use `kill -3` or `<Ctrl>-<Break>` to create the necessary thread dumps to diagnose your problem. Make sure to do this several times on each server, spaced about 5-10 seconds apart, to help diagnose deadlocks.
6. Compress the log file using the below UNIX utility or zip it using a Windows utility.

   ```
   % tar czf logfile.tar logfile.txt
   ```
7. *Attach* the compressed log file to an e-mail to your Oracle Support representative. Do not cut and paste the log file into the body of an e-mail.

Getting an Oracle HotSpot VM Thread Dump

If you use the Oracle HotSpot VM, use one of the following methods to generate a thread dump:

- Use the WLST `threadDUMP` command.
- Use the `jstack` utility.
- If you are using the Oracle HotSpot VM in Linux, use `Kill -3` *PID*, where *PID* is the root of the process tree.

  To obtain the root PID, run the below command.

  ```
  ps -efHl | grep 'java' **. **
  ```

  Using a `grep` argument that is a string will be found in the process stack that matches the server startup command. The first PID reported will be the root process, assuming that the `ps` command has not been piped to another routine.

  In Linux, each execute thread appears as a separate process under the Linux process stack. To use `Kill -3` on Linux, your supply must match the PID of the main WebLogic execute thread. Otherwise, no thread dump will be produced.
- If you are using the Oracle HotSpot VM in Windows, you can use the `Ctrl-Break` command on the application console to generate a thread dump.

Check Garbage Collection

If you are experiencing cluster problems, you should also check the garbage collection on the Managed Servers. If garbage collection is taking too long, the servers will not be able to make the frequent heartbeat signals that tell the other cluster members they are running and available.

If garbage collection (either first or second generation) is taking 10 or more seconds, you need to tune heap allocation (the `msmx` parameter) on your system.

Run utils.MulticastTest

You can verify that multicast is working by running `utils.MulticastTest` from one of the Managed Servers. For more information, see Using the Oracle WebLogic Server Java Utilities in *Command Reference for Oracle WebLogic Server*.

14

Troubleshooting Multicast Configuration

Read the suggestions for troubleshooting IP multicast configuration problems.

Using IP multicasting, Oracle WebLogic Server instances in a cluster can share a single IP address and port number. This capability enables all members of a cluster to be treated as a single entity and enables members of the cluster to communicate among themselves.

For general information on using and configuring multicast within a cluster, see Cluster Configuration and config.xml.

For general cluster troubleshooting suggestions, see Troubleshooting Common Problems.

This chapter includes the following sections:

Verifying Multicast Address and Port Configuration

The first step in troubleshooting multicast problems is to verify that you have configured the multicast address and port correctly. A multicast address must be correctly configured for each cluster.

Multicast address and port configuration problems are among the most common reasons why a cluster does not start or a server fails to join a cluster. The following considerations apply to multicast addresses:

- The multicast address must be an IP address between 224.0.0.0 and 239.255.255.255 or a host name with an IP address in this range.
- The default multicast address used by WebLogic Server is 239.192.0.0.
- Do not use any *x*.0.0.1 multicast address where *x* is between 0 and 9, inclusive.

Possible Errors

The following types of errors commonly occur due to multicast configuration problems:

- Unable to create a multicast socket for clustering
- Multicast socket send error
- Multicast socket receive error

Checking the Multicast Address and Port

To check the multicast address and port, do one of the following:

- Check the cluster multicast address and port through the WebLogic Remote Console.
- Check the multicast information of the `<cluster>` element in `config.xml`.

Identifying Network Configuration Problems

After you verify that the multicast address and port are configured correctly, determine whether network problems are interfering with multicast communication.

Physical Connections

Ensure that no physical problems exist in your network.

- Verify the network connection for each machine that hosts servers within the cluster.
- Verify that all components of the network, including routers and DNS servers, are connected and functioning correctly.

Address Conflicts

Address conflicts within a network can disrupt multicast communications.

- Use the `netstat` utility to verify that no other network resources are using the cluster multicast address.
- Verify that each machine has a unique IP address.

nsswitch.conf Settings on UNIX Systems

On UNIX systems, you may encounter the `UnkownHostExceptions` error. This error can occur multiple times, even when the server is not under a heavy load. Check `/etc/nsswitch.conf` and change the order to `'files,DNS,NIS'` to avoid this error.

See the `nsswitch.conf` man page for your system.

Using the MulticastTest Utility

After you verify that the multicast address and port are configured correctly and there are no physical or configuration problems with your network, you can use `utils.MulticastTest` to verify that multicast is working and to determine if unwanted traffic is occurring between different clusters.

For more information about the MulticastTest utility, see `MulticastTest` in *Command Reference for Oracle WebLogic Server.*

If MulticastTest fails and the machine is multihomed, ensure that the primary address is being used. See Multicast and Multihomed Machines.

> **Note**
>
> You should set `-Djava.net.preferIPv4Stack=true` when specifying an IPv4 format address for the multicast address on Linux machines running dual IPv4/IPv6 stacks.

Tuning Multicast Features

Learn how to tune various features of WebLogic Server to work with multicasting.

Multicast Timeouts

Multicast timeouts can occur during a Network Interface Card (NIC) failover. Timeouts can result in the following error message:

```
<Error><Cluster><Multicast socket receive error:
java.io.InterruptedIOException: Receive timed out>
```

When this error occurs, you can:

- Disable the NIC failover.
- Disable the `igmp snooping` switch. This switch is part of the Internet Group Management Protocol (IGMP) and is used to prevent multicast flood problems on the managed switch.
- On Windows platforms, check the IGMP level to ensure that multicast packets are supported.
- Set the Multicast TTL as follows:

  `MulticastTTL=32`
  For more information, see Configure Multicast Time-To-Live (TTL).

Cluster Heartbeats

Each WebLogic Server instance in a cluster uses multicast to broadcast regular heartbeat messages that advertise its availability. By monitoring heartbeat messages, server instances in a cluster determine when a server instance has failed.

The following sections describe possible solutions when cluster heartbeat problems occur.

Multicast Send Delay

Multicast Send Delay specifies the amount of time the server waits to send message fragments through multicast. This delay helps to avoid OS-level buffer overflow. This can be set using the `MulticastSendDelay` attribute of the Cluster MBean. See the *MBean Reference for Oracle WebLogic Server*.

Operating System Parameters

If problems still occur after setting the Multicast Send Delay, you may need to set the following operating system parameters related to UDP settings:

- `xdp_xmit_hiwat`
- `udp_recv_hiwat`

If these parameters are set to a lower value (for example, 8K), there may be a problem if the multicast packet size is set to the maximum allowed (32K). Try setting these parameters to 64K.

Multicast Storms

A multicast storm is the repeated transmission of multicast packets on a network. Multicast storms can stress the network and attached stations, potentially causing end-stations to hang or fail.

Increasing the size of the multicast buffers can improve the rate at which announcements are transmitted and received, and prevent multicast storms. See Configure Multicast Buffer Size.

Multicast and Multihomed Machines

The following considerations apply when using multicast in a multihomed environment:

- Ensure that you have configured a **Unix Machine** instance from the WebLogic Remote Console and have specified an `InterfaceAddress` for each server instance to handle multicast traffic.
- Run `/usr/sbin/ifconfig -a` to check the MAC address of each machine in the multihomed environment. Ensure that each machine has a unique MAC address. If machines use the same MAC address, this can cause multicast problems.

Multicast in Different Subnets

If multicast problems occur when cluster members are in different subnets you should configure Multicast-Time-To-Live. The value of the Multicast TTL parameter for the cluster must be high enough to ensure that routers do not discard multicast packets before they reach their final destination.

The Multicast TTL parameter sets the number of network hops that a multicast message makes before the packet can be discarded. Configuring the Multicast TTL parameter appropriately reduces the risk of losing the multicast messages that are transmitted among server instances in the cluster.

See Configure Multicast Time-To-Live (TTL).

Debugging Multicast

If you are still having problems with the multicast address after performing the troubleshooting tips, gather debugging information for multicast.

Debugging Utilities

The following utilities can help you debug multicast configuration problems.

MulticastMonitor

MulticastMontior is a standalone Java command line utility that monitors multicast traffic on a specific multicast address and port. The syntax for this command is:

```
java weblogic.cluster.MulticastMonitor <multicast_address> <multicast_port>
<domain_name> <cluster_name> <domain_directory>
```

MulticastTest

The MulticastTest utility helps you debug multicast problems when you configure a WebLogic cluster. The utility sends out multicast packets and returns information about how effectively multicast is working on your network.

Debugging Flags

The following debug flags are specific to multicast:

- `DebugCluster`
- `DebugClusterHeartBeats`
- `DebugClusterFragments`

See the following sections:

Setting Debug Flags on the Command Line

Set these flags from the command line during server startup by adding the following options:

- `-Dweblogic.debug.DebugCluster=true`
- `-Dweblogic.debug.DebugClusterHeartBeats=true`
- `-Dweblogic.debug.DebugClusterFragments=true`

Setting Debug Attributes Using WLST

Set debug attributes using the following WLST commands:

```
connect()
edit()
startEdit()
servers=cmo.getServers()
for s in servers:
  d=s.getServerDebug()
  d.setDebugCluster(true)
activate()
```

Miscellaneous Issues

Consider miscellaneous multicast issues you may encounter.

File Descriptor Problems

Depending on the operating system, there may be problems with the number of file descriptors open. On UNIX, you can use `loses` to determine how many files on disk a process has open. If a problem occurs, you may need to increase the number of file descriptors on the machine.

Other Resources for Troubleshooting Multicast Configuration

When resolving multicast problems, certain resources may be helpful.

- *Oracle Fusion Middleware Release Notes for Microsoft Windows*
- Oracle Support: https://support.oracle.com/
- Oracle Forums: http://forums.oracle.com/

A
The WebLogic Cluster API

Learn how to use the Oracle WebLogic Server Cluster API.

This appendix includes the following sections:

How to Use the API

The WebLogic Cluster public API is contained in a single interface, `weblogic.rmi.cluster.CallRouter`.

```
Class java.lang.Object
   Interface weblogic.rmi.cluster.CallRouter
      (extends java.io.Serializable)
```

A class implementing this interface must be provided to the RMI compiler (`rmic`) to enable parameter-based routing. Run `rmic` on the service implementation using these options (to be entered on one line):

```
$ java weblogic.rmic -clusterable -callRouter
    <callRouterClass> <remoteObjectClass>
```

The call router is called by the clusterable stub each time a remote method is invoked. The router is responsible for returning the name of the server to which the call should be routed.

Each server in the cluster is uniquely identified by its name as defined with the WebLogic Remote Console. These are the names that the method router must use for identifying servers.

Example: Consider the `ExampleImpl` class which implements a remote interface Example, with one method `foo`:

```
public class ExampleImpl implements Example {
  public void foo(String arg) { return arg; }
}
```

This `CallRouter` implementation `ExampleRouter` ensures that all `foo` calls with `'arg' < "n"` go to server1 (or server3 if server1 is unreachable) and that all calls with `'arg' >= "n"` go to server2 (or server3 if server2 is unreachable).

```
public class ExampleRouter implements CallRouter {
  private static final String[] aToM = { "server1", "server3" };
  private static final String[] nToZ = { "server2", "server3" };

  public String[] getServerList(Method m, Object[] params) {
    if (m.GetName().equals("foo")) {
      if (((String)params[0]).charAt(0) < 'n') {
        return aToM;
      } else {
        return nToZ;
      }
    } else {
      return null;
    }
  }
}
```

This `rmic` call associates the `ExampleRouter` with `ExampleImpl` to enable parameter-based routing:

```
$ rmic -clusterable -callRouter ExampleRouter ExampleImpl
```

Custom Call Routing and Collocation Optimization

If a replica is available on the same server instance as the object calling it, the call will not be load balanced, because it is more efficient to use the local replica.

See Optimization for Collocated Objects.

Index