Oracle WebLogic Server 12c: Administration I

Student Guide – Volume 2

D80149GC11
Edition 1.1
February 2014
D85488

ORACLE

Authors
Bill Bell
Elio Bonazzi
TJ Palazzolo
Steve Friedberg

Technical Contributors and Reviewers
Mark Lindros
Will Lyons
Serge Moiseev
Matthew Slingsby
Angelika Krupp
Kevin Tate
Takyiu Liu
Jenny Wongtangswad
Juan Quezada
Radek Felcman
Matt Heimer
Saskia Nehls
Juan Adauco Quezada

Editors
Aju Kumar
Malavika Jinka

Graphic
Seema Bopaiah

Publishers
Giri Venugopal
Jayanthy Keshavamurthy

Contents

11

Network Channels and Virtual Hosts

Objectives

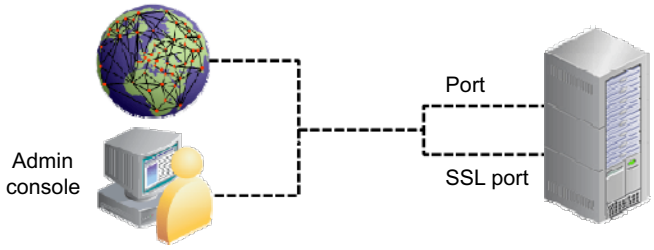
After completing this lesson, you should be able to configure:

- A WebLogic Server network channel
- WebLogic Server to use an administration port
- A virtual host for WebLogic Server

Default WebLogic Networking

By default, an instance of WebLogic Server binds:

- To all available network interfaces on the host machine
- To a single port
- To a separate port for secure sockets layer (SSL) communication (if configured)



For many development environments, configuring WebLogic Server network resources is simply a matter of identifying a server's listen address, listen port, and optionally an SSL port. You can configure this address, port, and SSL port by using the server's **Configuration** > **General** page in the administration console.

If your server instance runs on a multihomed machine and you do not configure a listen address, the server binds the listen port and/or SSL listen ports to all available IP addresses on the multihomed machine.

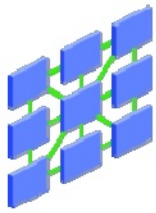
This default configuration may meet your needs if:

- Your application infrastructure has simple network requirements, such as during development or test
- Your server uses a single network interface, and the default port numbers provide enough flexibility for segmenting network traffic in your domain

Default WebLogic Networking

The default server port:

- Accepts all protocols (HTTP, T3, IIOP, SNMP)
- Supports various security and tuning parameters
- Is used for client-server communication
- Is used for remote server management (admin console, WLST)
- Is used for internal server-server communication:
  - Server startup and management messages
  - Cluster messages



You can configure each WebLogic Server instance to communicate over a variety of protocols, such as Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), Internet Inter-ORB Protocol (IIOP), the WebLogic Server proprietary protocol called T3, and the Simple Network Management Protocol (SNMP). In addition, you can configure general server network settings that apply to all protocols. The default listen address and port accept all types of incoming server communications, including:

- Web application HTTP requests
- Remote access to the server Java Naming and Directory Interface (JNDI) tree
- Remote Enterprise JavaBeans (EJB) application Remote Method Invocations (RMI)
- Simple Network Management Protocol (SNMP) polling requests
- Configuration and monitoring requests from remote management tools, such as the admin console or WLST
- Configuration and monitoring requests sent from the administration server to the managed server
- Initial startup messages sent from a managed server to the administration server
- Messages sent between cluster members, such as for session replication

Additional Networking Scenarios

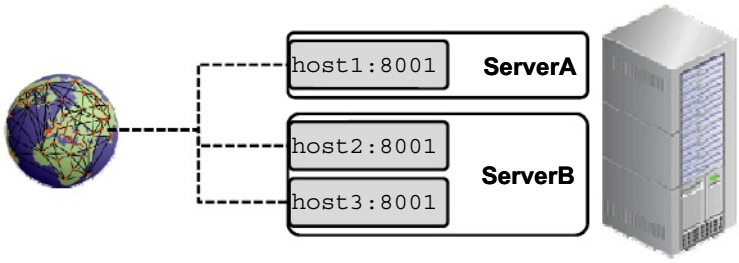
You can customize the default WebLogic Server network configuration to handle requirements such as:

- Dedicating network interfaces to specific servers
- Using multiple ports on a single server
- Isolating administrative communication
- Isolating internal cluster communication

In most production environments, administrators must balance finite network resources against the demands placed on the network. The task of keeping applications available and responsive can be complicated by specific application requirements, security considerations, and maintenance tasks, both planned and unplanned. WebLogic Server allows you to control the network traffic associated with your applications in a variety of ways and configure your environment to meet the varied requirements of your applications and end users.

Dedicating Network Interfaces to Specific Servers

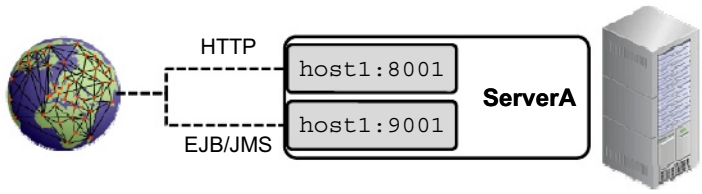
- You can dedicate specific network interfaces to multiple WebLogic Server instances running on the same machine.
- The characteristics for each network interface is set at the operating system level.



A typical production machine hosts multiple WebLogic Server instances and also is installed with multiple network interface cards (NICs). In this scenario, it may be desirable to explicitly associate each server with its own dedicated interface or interfaces. For example, each NIC could then be tuned to support different levels of performance to match the expected load on the assigned server instance. This approach also gives administrators the flexibility to bind multiple servers on the same machine to the same port number.

Using Multiple Ports on a Single Server

- You can dedicate specific ports for certain client protocols. This allows you to:
  - Configure different security, performance, or quality of service characteristics for each port
  - Enable and disable ports independently of one another

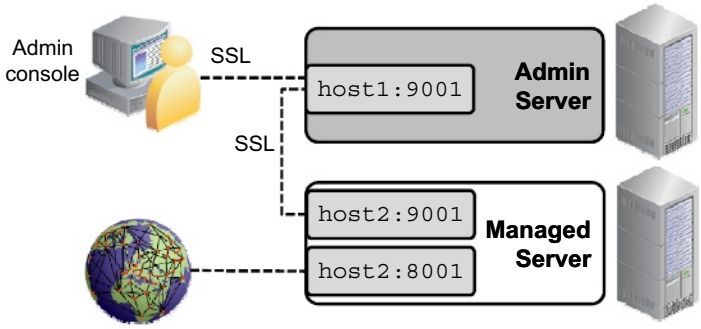


ORACLE

WebLogic Server allows a single server instance to bind to multiple ports and allows administrators to define different network settings for each port. By using this approach, you can dedicate each port to a different protocol, such as HTTP(S) or T3(S). Also, because each port can be brought up and down independently, administrators gain additional flexibility in how they can perform server maintenance.

Isolating Administrative Communication

- Create a dedicated address and/or port for administrative traffic.
  - You can disable client access while retaining administrative access for server maintenance or troubleshooting.



While maintaining or troubleshooting a production server, it is often desirable to disable all incoming application requests. However, a server's default network configuration implies that all traffic run on the same port. Therefore, if the port were closed, all remote administration tools such as the admin console or WLST would also not be able to connect to the server.

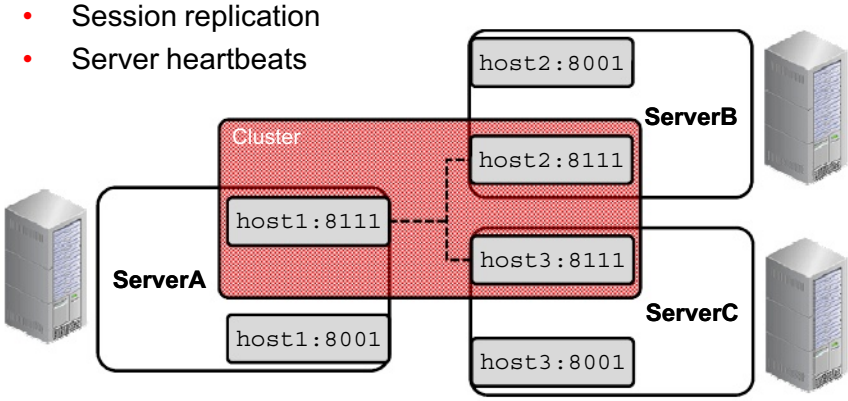
WebLogic Server supports a domain in which all servers use a separate SSL port that accepts only administration requests. The use of dedicated administration ports enables you to:

- **Start a server in standby state:** This allows you to administer a server, whereas its other network connections are unavailable to accept client connections.
- **Separate administration traffic from application traffic in your domain:** In production environments, separating the two forms of traffic ensures that critical administration operations (starting and stopping servers, changing a server's configuration, and deploying applications) do not compete with high-volume application traffic on the same network connection.
- **Administer a deadlocked server instance:** If you do not configure an administration port, administrative commands such as THREAD_DUMP and SHUTDOWN will not work on deadlocked server instances.

Isolating Cluster Communication

Use a dedicated address and/or port for peer-to-peer and one-to-many cluster messaging:

- Session replication
- Server heartbeats



ORACLE

Similar to administration ports, the servers within a cluster can also use separate ports dedicated to internal cluster communication. Administrators have the option to configure these clusters or "replication" ports to use either a standard or a secure (SSL) protocol.

In general, ports on a server that can be used to send internal messages to other servers in the same domain are called "outgoing" ports. Ports that are not enabled for "outgoing" are used solely to process incoming client requests.

Network Channel

- A network channel consists of:
  - A listen address and port
  - A single supported protocol along with its service characteristics

- Each server:
  - Has an implicit default channel, which can be disabled
  - Has a default SSL channel (if configured)
  - Supports all protocols by default
  - Can be assigned additional channels
- Channels can be created, enabled, or disabled dynamically without restarting the server.

ORACLE

A network channel is a configurable resource that defines the attributes of a network connection for a specific WebLogic Server instance. A network channel definition includes a listen address, port number, supported protocol, and whether or not it can be used for internal server-to-server communication. You can use network channels to manage quality of service, meet varying connection requirements, and improve the utilization of your systems and network resources.

Administrators create a channel for a specific server instance. Channels are not defined globally and applied to one or more servers. You can assign multiple channels to a server instance, but each channel must have a unique combination of listen address, listen port, and protocol. Similarly, if you assign non-SSL and SSL channels to the same server instance, make sure that they do not use the same port number.

If you want to disable the non-SSL listen port so that the server listens only on the SSL listen port, deselect Listen Port Enabled in the Configuration > General settings for the server. Similarly, if you want to disable the SSL listen port so that the server listens only on the non-SSL listen port, deselect SSL Listen Port Enabled. Note that unless you define custom network channels, you cannot disable both the default non-SSL listen port and the SSL listen port. At least one port must be active on a server.

Channel Selection

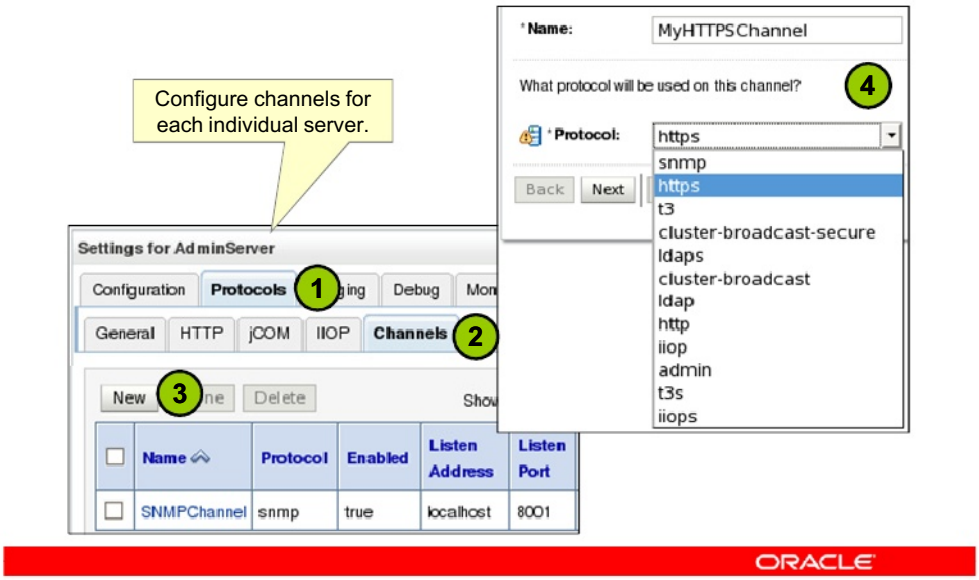
- For internal communication, WebLogic Server tries to select an available channel that best fits the requirements (supports the protocol).
- Examples include:
  - Monitoring a managed server from the administration server
  - Sending a cluster replication message
  - Accessing the embedded Lightweight Directory Access Protocol (LDAP)
- If multiple channels meet the criteria, messages are distributed across them:
  - Evenly (default)
  - Using channel weights

ORACLE

If you have not created a custom network channel to handle outgoing administrative or clustered traffic, WebLogic Server uses the default channel. If instead, multiple channels exist for these internal protocols, the server will evenly distribute the outgoing messages across the available channels. Administrators also have the option of assigning numeric weights (1–100) to internal channels for situations in which the load should not be evenly distributed.

When initiating a connection to a remote server, and multiple channels with the same required destination, protocol, and quality of service exist, WebLogic Server tries each in turn until it successfully establishes a connection or runs out of channels to try.

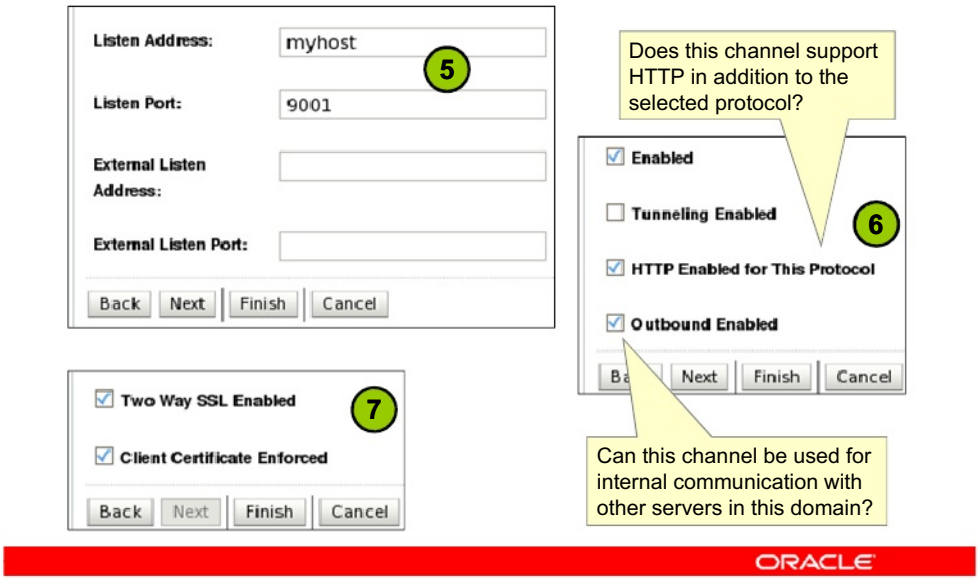
Creating a Channel



To configure a network channel for an existing server, perform the following steps:

1. After locking the configuration, select the server, and then click its **Protocols** tab.
2. Click the **Channels** subtab.
3. Click **New**.
4. Enter a name for the channel, select the protocol it will accept or use, and click **Next**. For administrative channels, select the admin protocol. For cluster channels, select the “cluster-broadcast” or “cluster-broadcast-secure” protocol.

Creating a Channel



5. Enter a listen address and listen port that this channel binds to, and click **Next**. If an address is not supplied, the address of the default channel is used.
6. Select the check boxes you want and click **Next**. By default, the new channel will be enabled and automatically bind to its address and port. If instead you want to enable it manually at a later time, deselect the **Enabled** check box. Other options include:
   - **Tunneling Enabled:** Select this to enable HTTP tunneling. It is disabled by default. Under the HTTP protocol, a client may only make a request, and then accept a reply from a server. The server may not voluntarily communicate with the client, and the protocol is stateless, meaning that a continuous two-way connection is not possible. WebLogic HTTP tunneling simulates a T3 connection via the HTTP protocol, overcoming these limitations. Note that the server must also support both the HTTP and T3 protocols to use HTTP tunneling.
   - **HTTP Enabled for This Protocol:** Specifies whether HTTP traffic should be allowed over this network channel. HTTP is generally required with other binary protocols for downloading stubs and other resources. Note that this is only applicable if the selected protocol is not HTTP or HTTPS.

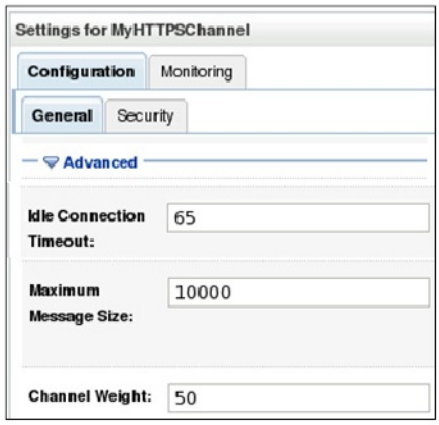
- **Outbound Enabled:** Specifies whether new server-to-server connections can consider this network channel when initiating a connection. Leave this field deselected for client channels.

7. For secure protocols, optionally enable two-way SSL. Click **Finish**.

If your server is being accessed through a proxy server on a separate listen address and/or port, you may be required to supply an external listen address and/or external listen port for each channel. These values are used in cases where the server must publish its location to external clients, such as a web server plug-in or a hyperlink in a web browser.

Channel Network Settings

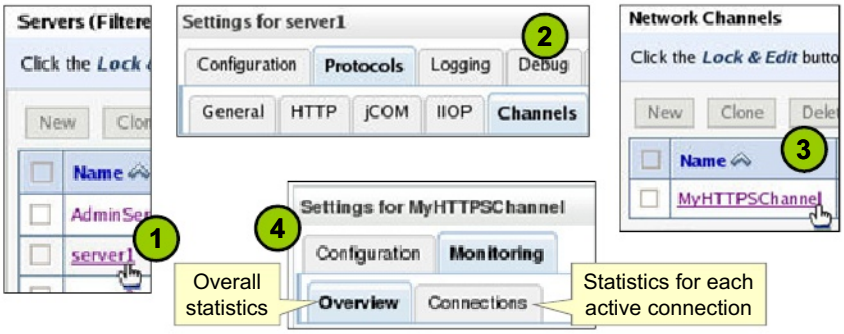
Custom channels inherit their network settings from the default channel, if not overridden.



Both the default and custom network channels support various general and protocol-specific network settings. If not configured, custom channels inherit their network settings from the default channel. These settings include:

- **Cluster Address:** The address this network channel uses to generate cluster-aware EJB stubs for load balancing and failover in a cluster
- **Accept Backlog:** The number of backlogged, new TCP connection requests that this network channel allows
- **Maximum Backoff Between Failures:** The maximum wait time (in seconds) between failures while accepting client connections
- **Idle Connection Timeout:** The maximum amount of time (in seconds) that a connection is allowed to be idle before it is closed by this network channel. This timeout helps guard against server deadlock through too many open connections.
- **Maximum Message Size:** This maximum attempts to prevent a denial of service attack whereby a caller sends very large messages, thereby keeping the server from responding quickly to other requests
- **Channel Weight:** A weight to give this channel when multiple channels are available for internal server-to-server connections

Monitoring Channels



Runtime statistics are available for each channel:

- Number of active connections
- Messages sent/received
- Bytes sent/received

1. Select a server.
2. Select **Protocols** > **Channels**.
3. Select a channel.
4. Click the **Monitoring** tab.

Administration Port

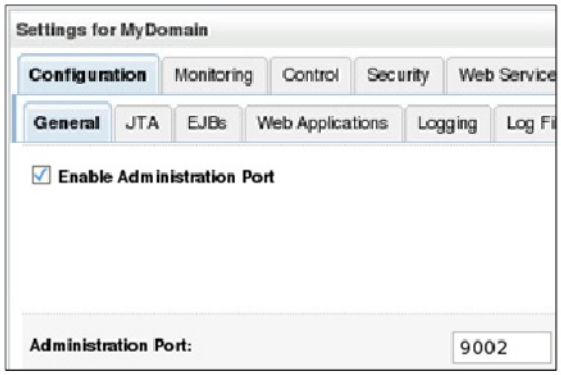
- The domain's *Administration Port* is an alternative to creating a custom network channel with the explicit "admin" protocol on every server.
- This feature creates an implicit administration channel on every server using:
  - The listen address of the server's default channel
  - The same admin port number (entered by you) for each server
- Administration channels require SSL to be configured on each server.

You can define an optional administration port for your domain. When configured, the administration port is used by each managed server in the domain exclusively for communication with the domain's administration server. If an administration port is enabled, WebLogic Server automatically generates an administration channel based on the port settings upon server instance startup.

The administration port accepts only secure, SSL traffic, and all connections via the port require authentication. The Administration Server and all managed servers in your domain must be configured with support for the SSL protocol. Managed servers that do not support SSL cannot connect with the administration server during startup. You will have to disable the administration port to configure these managed servers.

Ensure that each server instance in the domain has a configured default listen port or default SSL listen port. The default ports are those you assign on the **Server** > **Configuration** > **General** page in the administration console. A default port is required in the event that the server cannot bind to its configured administration port. If an additional default port is available, the server continues to boot and you can change the administration port to an acceptable value.

Configure the Domain's Administration Port



To use a domain-wide administration port, perform the following steps:

1. Select the domain name in the left panel.
2. On the default **Configuration** > **General** tab, select the **Enable Administration Port** check box.
3. Enter a value for the **Administration Port** field and click **Save**.

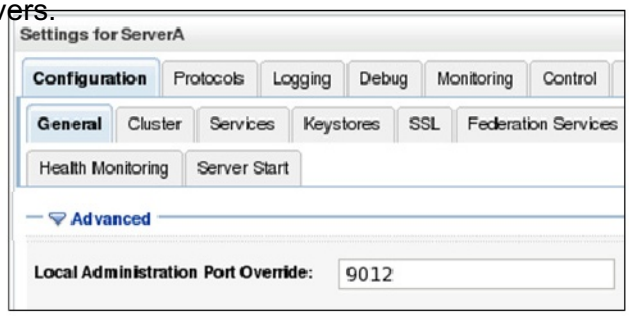
After enabling the administration port, all administration console and WLST traffic must connect via the administration port.

If you boot managed servers either at the command line or by using a start script, specify the administration port in the administration server's URL. The URL must also specify the HTTPS protocol rather than HTTP. If you use Node Manager for starting managed servers, it is not necessary to modify startup settings or arguments for the managed servers. Node Manager obtains and uses the correct URL to start a managed server.

Server Override of the Administration Port

If multiple servers run on the same machine, you must perform one of the following:

- Bind each server to a unique network address.
- Override the administration port number for individual servers.

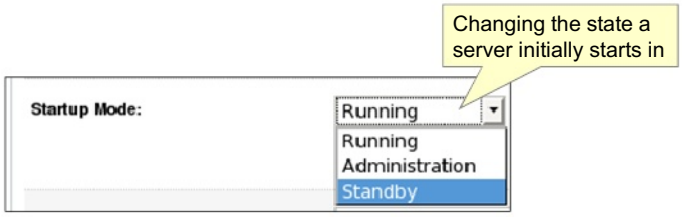


ORACLE

Override the domain-wide port on all but one of the server instances running on the same machine. Override the port by using the Local Administration Port Override option on the **Advanced** portion of the server's **Configuration** > **General** tab in the administration console.

Server Standby Mode

- In the ADMIN state, all channels are opened but applications are only accessible to administrators.
- In the STANDBY state, only a server's administration channel is opened. All other channels remain closed.
- Administrators can later transition servers in the ADMIN or STANDBY state to the RUNNING state.

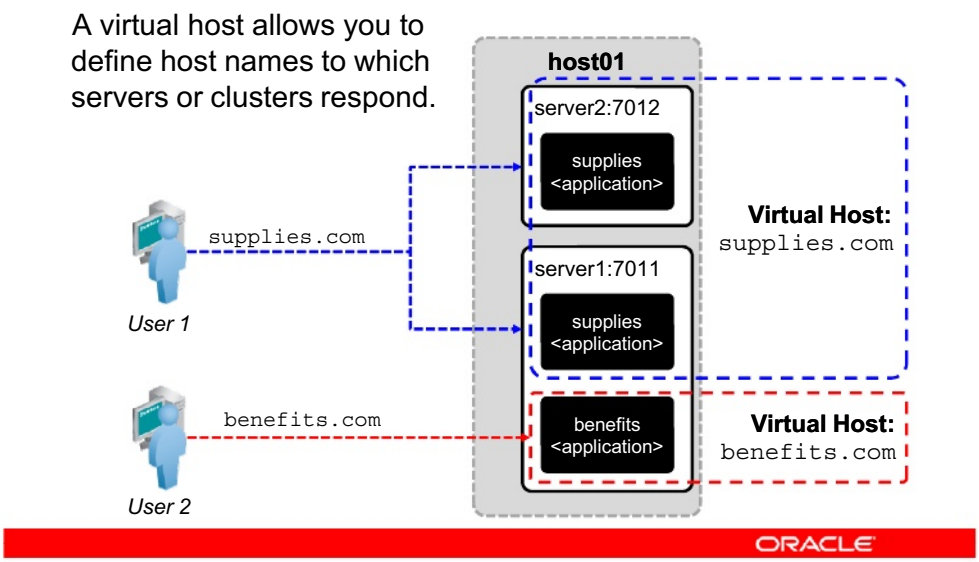


In the ADMIN state, WebLogic Server is up and running, but available only for administration operations, allowing you to perform server-level and application-level administration tasks. The server instance accepts requests from users with the *Admin* role. Requests from non-admin users are refused. All applications accept requests from users with the *Admin* and *AppTester* roles. The Java Database Connectivity (JDBC), Java Message Service (JMS), and Java Transaction API (JTA) subsystems are active, and administrative operations can be performed upon them. However, you do not have to have administrator-level privileges to access these subsystems when the server is in the ADMIN state.

A server instance in STANDBY does not process any client requests. All nonadministration network channels do not open. The server's administration channel is open and accepts lifecycle commands that transition the server instance to either the RUNNING or the SHUTDOWN state. Other types of administration requests are not accepted. Starting a server instance in standby is a method of keeping it available as a "hot" backup, a useful capability in high-availability environments.

The only way to cause a server instance to enter the STANDBY state and remain in that state is through the server's Startup Mode attribute, found in the **Advanced** section of the **Configuration** > **General** tab.

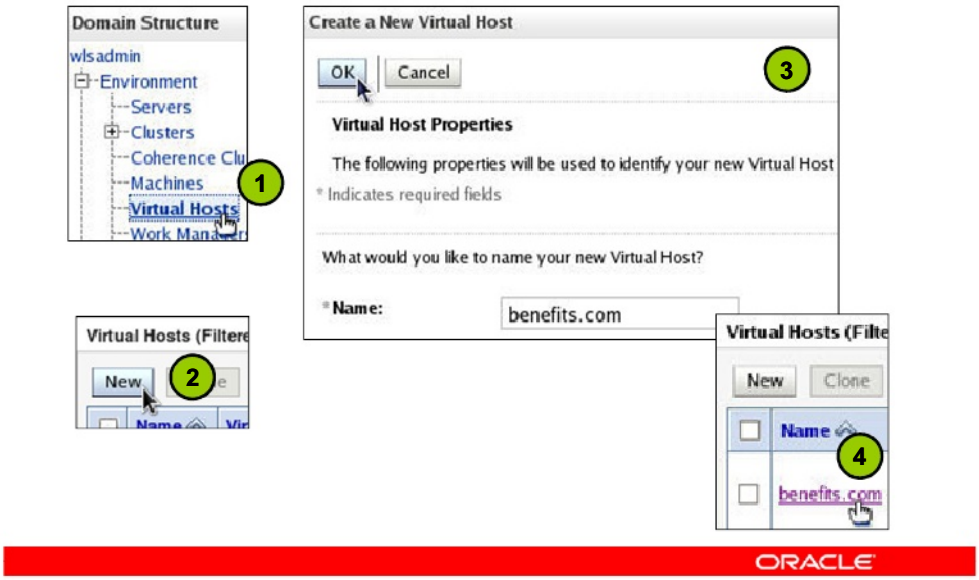
Virtual Host



A virtual host allows you to use DNS to map a host name to an IP address of an instance of WebLogic Server or a cluster and specify which servers running on that IP address are part of the virtual host. This allows you to target an application to a virtual host during deployment, which in turn determines which servers ultimately host the application. Update DNS to map the host name to the correct IP address. (You can also use the `hosts` file to do this.) When WebLogic receives requests for that host name, it matches it with the appropriate virtual host and sends the request to the correct instance of WebLogic Server.

WebLogic also allows you to configure separate HTTP parameters, access logs, and default web applications for each virtual host. Defining a default web application for a virtual host allows you to configure multiple applications on the same domain or servers that are each accessible without specifying a web application's context path. Instead, the host name of the URL determines which application gets called.

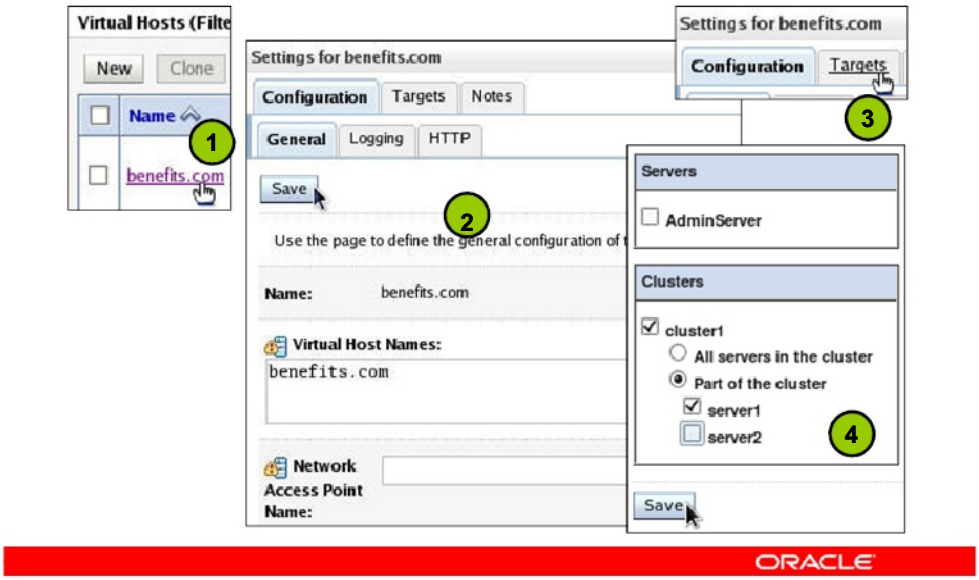
Create a Virtual Host



You perform the following steps to create a virtual host using the administration console:

1. Within the Domain Structure, you expand the **Environment** node and select **Virtual Hosts**.
2. Click **New**.
3. Enter a unique name for your virtual host. Note that this does not have to match the host name you configure. Click **OK**.
4. Verify that your virtual host is created.

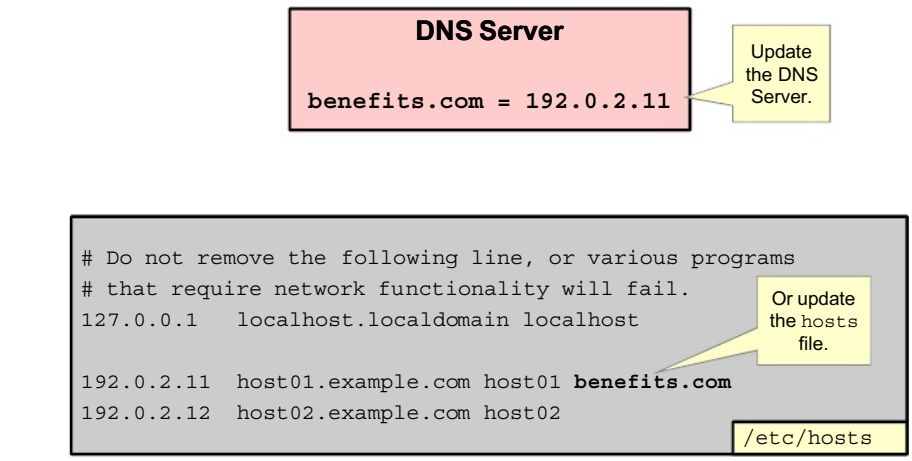
Configure a Virtual Host



You perform the following steps to configure your virtual host using the WebLogic administration console:

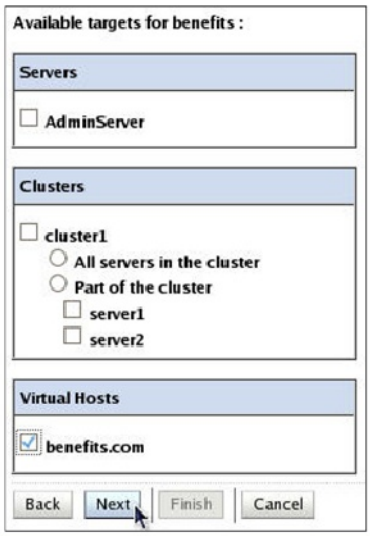
1. Select your newly created virtual host in the virtual host list.
2. Enter the host names that correspond to this virtual host. In this case, a virtual host called `benefits.com` is added. You can potentially configure a separate network channel and specify the name of that channel in the Network Access Point Name field. In this example, this field is left blank so the virtual host is associated with the default channel. Click **Save** when finished.
3. Select the **Targets** tab to specify which servers in the domain are associated with your virtual host.
4. In this case, `server1` is selected. Click **Save** and activate your changes.

Configure a Virtual Host in DNS or the `hosts` File



To map your host name to the IP address on which the server is listening either modify your DNS server or modify the machine's */etc/hosts* file.

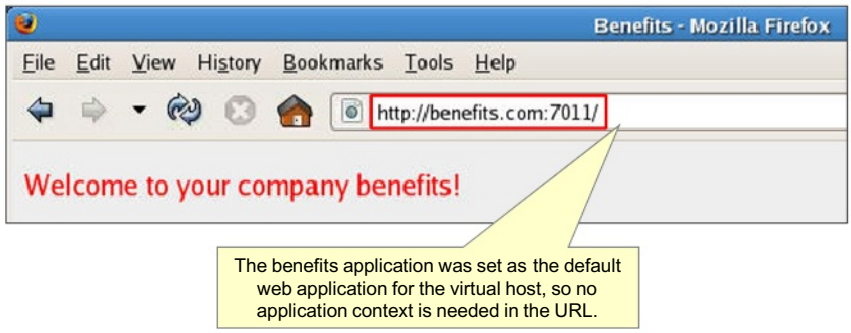
Deploy to a Virtual Host



Now that DNS is mapped to the server's IP address and your virtual host is configured, you deploy the application to your virtual host. You deploy the application as usual but when you arrive at the targeting page there is a new category called Virtual Hosts. You select your virtual host and continue as usual with the rest of the deployment process.

Run the Application Using the Virtual Host

`http://benefits.com:7011`



The benefits application was set as the default web application for the virtual host, so no application context is needed in the URL.

After everything is configured and deployed, you test your virtual host connection by accessing your application using the virtual host name rather than the server's IP address. In this example, instead of the IP address, the URL contains `benefits.com`, followed by the server's port number.

Quiz

Which one is not a protocol that you can assign to a network channel?

a. ADMIN
b. T3
c. HTTPS
d. ORCL

Answer: d

Quiz

Setting up a domain-wide admin port *requires* that all servers in the domain:

a. Use the same port number for it
b. Have SSL configured
c. Be changed to start in STANDBY or ADMIN mode

Answer: b

Summary

In this lesson, you should have learned how to configure:

- A WebLogic Server network channel
- WebLogic Server to use an administration port
- A virtual host for WebLogic Server

Practice 11-1 Overview: Configuring a Network Channel

This practice covers the following topics:

- Configuring a custom network channel for HTTP traffic
- Accessing an application by using the custom channel

Practice 11-2 Overview: Configuring the Administration Port

This practice covers the following topics:

- Configuring the domain-wide admin port
- Accessing the administration server by using the admin port
- Starting an application in Admin mode
- Accessing an application in Admin mode for testing

Practice 11-3 Overview: Creating a Virtual Host

This practice covers the following topics:

- Creating a virtual host
- Deploying an application to the virtual host
- Accessing an application through the virtual host

12 Clusters

Overview, Creation, and Configuration

Objectives

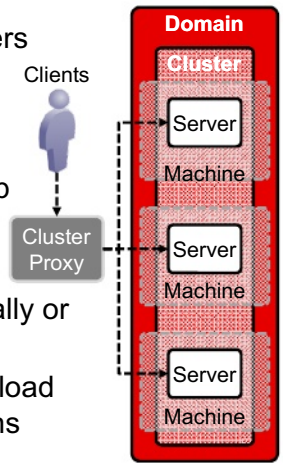
After completing this lesson, you should be able to:

- Describe two cluster architectures: basic and multi-tier
- Create and configure a cluster
- Create and configure a dynamic cluster

Cluster: Review

A cluster:

- Is a logical group of managed servers from the same domain that run cooperatively
- Supports features that provide high availability for web applications, web services, EJBs, and JMS
- Is transparent to its clients
- Can have servers added to it statically or dynamically
- Requires a cluster proxy to provide load balancing, if it hosts web applications



A WebLogic Server cluster consists of one or more managed servers from the same domain running simultaneously and working together to provide increased reliability and scalability. A cluster appears to clients as one WebLogic Server instance. The server instances that constitute a cluster can run on one machine or on multiple machines.

A cluster achieves high availability through the replication of services. Because of this replication, failover is possible. When one server fails, a second server automatically can resume operation where the first server left off.

Load balancing, the distribution of work across the cluster, ensures that each server in the cluster helps carry the load.

Scalability is achieved because you can increase a cluster's capacity by adding server instances to the cluster, without making any architectural changes.

A cluster also assists in migration. After a system failure on one server, work can be continued by moving the services that server provided to another server in the cluster (service level migration), or by moving the entire server to a new hardware (whole server migration).
After a cluster is created, configured servers can be added to it. A dynamic cluster is based on a server template. A server template sets server attributes. After a server template is assigned to a cluster, servers based on the template are generated and added to the cluster.

Clusters support different types of applications as follows:

- Web applications: Load balancing is achieved through the cluster proxy. This proxy can be a web server using a WebLogic Server proxy plug-in or a hardware load balancer. Failover is achieved by replicating or storing the HTTP session state of clients.
- For Enterprise JavaBeans (EJBs), clustering uses the EJB's replica-aware stub for load balancing and failover. 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.
- For JMS applications, clustering supports transparent access to distributed destinations from any member of the cluster.

Benefits of Clustering

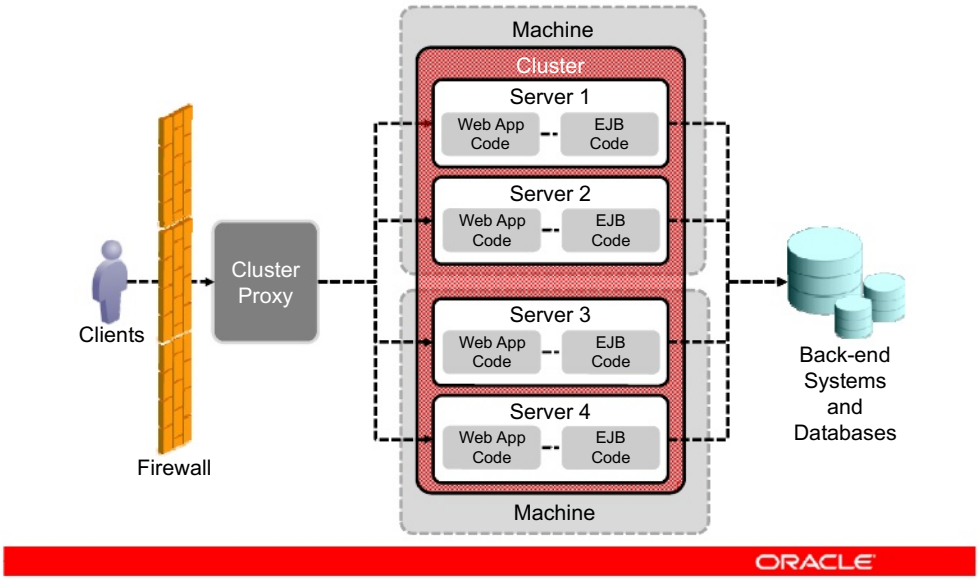
Concept	Description
Scalability	More capacity for applications can be provided by adding servers, without interruption of service or making architectural changes.
Load balancing	Work (for example, client requests) is distributed across the members of a cluster.
Failover	When a server fails, another one can automatically take its place. Information on the failed server is replicated (or stored), so that the new server has access to it.
Migration	When a server fails, its "pinned" services can continue by moving them to another server in the cluster, or by moving the entire failed server to a new hardware.

A "pinned" service is a service that must run only on a single instance of WebLogic Server at any given time.

A WebLogic Server cluster provides the following benefits:

- **Scalability:** The capacity of a cluster is not limited to one server or one machine. Servers can be added to the cluster dynamically to increase capacity. If more hardware is needed, a new server on a new machine can be added.
- **Load Balancing:** The distribution of jobs across the cluster members can be balanced, so no one server is overloaded.
- **Failover:** Distribution of applications and their objects on multiple servers enables easier failover of the session-enabled applications.
- **Availability:** A cluster uses the redundancy of multiple servers to insulate clients from failures. If one server fails, another can take over. With the replication (or storage) of server-specific information, the failover can be transparent to the client.
- **Migration:** This ensures uninterrupted availability of pinned services or components—those that must run only on a single server instance at any given time. An example of a pinned service is the use of the Java Transaction API (JTA).

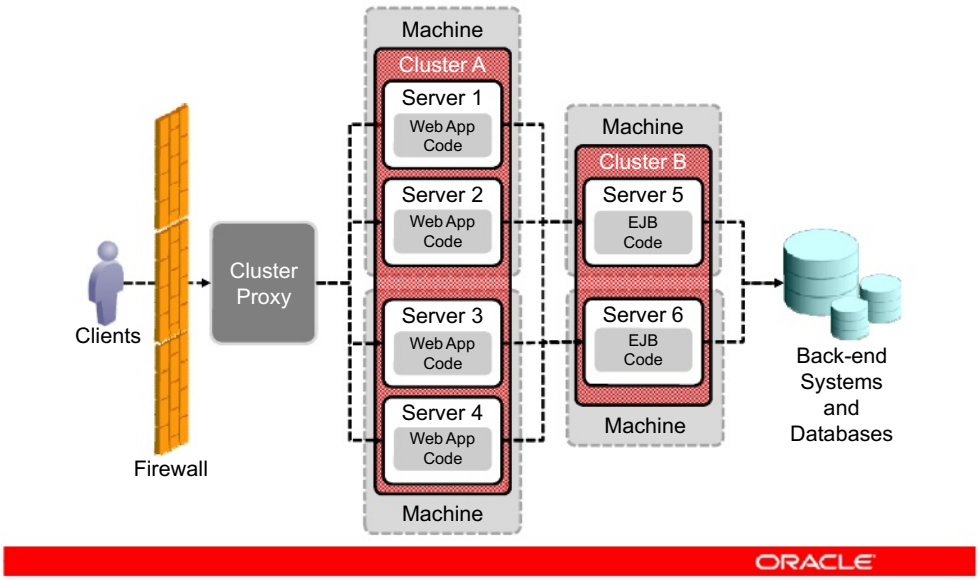
Basic (Single-Tier) Cluster Architecture



The basic, single-tier cluster architecture has all WebLogic Server application code in a single tier. That single tier includes both web applications and Enterprise JavaBeans.

Remove the “EJB Code” box for systems that do not use EJBs.

Multi-Tier Cluster Architecture



In the multi-tier cluster architecture, two separate WebLogic Server clusters are configured:

- Cluster A for the web application tier
- Cluster B to serve clustered EJBs

If your system does not use EJBs, you would not use the multi-tier cluster architecture in this way. You could have a second tier for JMS and use it to load balance JMS calls, however.

Architecture Advantages and Disadvantages

Cluster Architecture	Advantages	Disadvantages
Basic (single-tier)	• Easier to administer • Less network traffic • EJB calls are local (and therefore faster)	• Cannot load balance EJB calls
Multi-tier	• EJB calls are load balanced • Scaling options (for example, you can shift (or add) hardware and WebLogic server instances to whichever tier is busier) • More security options (for example, you could place a firewall in between the web application tier and the EJB tier)	• Harder to administer • Perhaps more hardware and licensing costs • EJB calls are remote (and therefore slower) • More network traffic

ORACLE

Basic (single-tier) advantages:

- Easier administration: There is only one cluster to create, configure, and maintain. Also, because one cluster hosts web applications and EJBs, you can easily deploy enterprise applications to the cluster. The web application and EJBs are in the same archive.
- Less network traffic: Clients access WebLogic Server through web applications. Calls from WebLogic Server to back-end systems and databases still occur, but all calls from the web application tier to EJBs are within the same JVM.
- Faster EJB performance: Because the calls from the web applications to the EJBs are within the same instance of WebLogic Server, there is no network overhead. The calls to the EJBs are local, not remote. This is especially important if the web applications make frequent EJB calls.

Basic (single-tier) disadvantage:

- EJB calls cannot be load balanced. Each call from the web application tier to an EJB is always to the EJB running on that same instance of WebLogic Server. It is therefore possible that the server load becomes unbalanced. Let us say that 200 concurrent users are accessing your applications, and they have been load balanced so that 50 are accessing each of the four servers in the cluster. It just so happens that the 50 users on server 1 are performing tasks that call EJBs, while the other 150 users on the other servers are not. Server 1 will be exceptionally busy, while servers 2, 3, and 4 will not. If the EJB calls were load balanced to an EJB tier, however, then the "EJB load" would be spread across all of the servers in the EJB tier cluster.

Multi-tier advantages:

- EJB calls are load balanced. Each call to an EJB can be load balanced across all the servers in the EJB cluster. The "unbalanced" situation described above is no longer possible.
- More scaling options: Separating the web application and EJB tiers onto separate clusters provides you with more options for scaling the system and distributing the load. For example, if users spend most of their time using the web applications, and those applications make few EJB calls, you can use a larger number of WebLogic Server instances in the web application cluster. If things change, and your applications become more EJB-intensive, you can shift or add servers to the EJB cluster.
- More security options: With another layer there is an opportunity to add more security. For example, you could place a firewall in between the web application and EJB clusters.

Multi-tier disadvantages:

- More difficult administration: There is more than one cluster to create, configure, and maintain. Also, because one cluster hosts web applications and the other EJBs, deployment becomes more complicated.
- Perhaps higher costs: With two clusters you may have more instances of WebLogic Server and more hardware.
- Slower EJB performance: Because the calls from the web applications to the EJBs are always remote, you must pay the price of remote calls. The applications must be developed with this in mind so that calls to EJBs are infrequent and "course grained."
- More network traffic: Because all EJB calls are remote, network traffic increases.

Cluster Communication

- Cluster members communicate with each other in two ways:
  - One-to-many messages:
    - For periodic “heartbeats” to indicate continued availability
    - To announce the availability of clustered services
    - **Note:** This communication can use either:
      - IP unicast (recommended): No additional configuration is required.
      - IP multicast: A multicast host and port must be configured.
  - Peer-to-peer messages:
    - For replicating HTTP session and stateful session EJB state
    - To access clustered objects that reside on a remote server (multi-tier architecture)
    - **Note:** This communication uses sockets.

An instance of WebLogic Server uses one-to-many communication to send regular “heartbeat” messages that advertise its continued availability to other server instances in the cluster. The servers in a cluster listen for heartbeat messages to determine when a server has failed.

All servers use one-to-many messages to announce the availability of clustered objects that are deployed or removed locally. Servers monitor these announcements so that they can update their local JNDI tree to indicate the current deployment of clustered objects. This is the maintenance of the so-called “cluster-wide” JNDI tree.

IP multicast enables multiple applications to subscribe to an IP address and port number, and listen for messages. A multicast address is an IP address in the range `224.0.0.0-239.255.255.255`. IP multicast does not guarantee that messages are received, so WebLogic Server allows for the possibility that some messages may be missed. If you use multicast, you must ensure your network propagates multicast messages to all clustered servers. The multicast time-to-live value can be increased if you find that messages are being missed. With multicast, you must ensure that no other applications share the multicast address and port, or servers will have to process extra messages, which introduces extra overhead.

Firewalls can break multicast transmissions. Although it might be possible to tunnel multicast transmissions through a firewall, this practice is not recommended. A final worry with multicast messaging is the possibility of a multicast "storm," in which server instances do not process incoming messages in a timely fashion, which leads to retransmissions and increased network traffic.

IP unicast is recommended because it does not have the network issues of multicast. You can set up a separate network channel for unicast communication, but it is not required. If no separate channel is defined, each server's default channel is used (the default channel is the server's configured host and port).

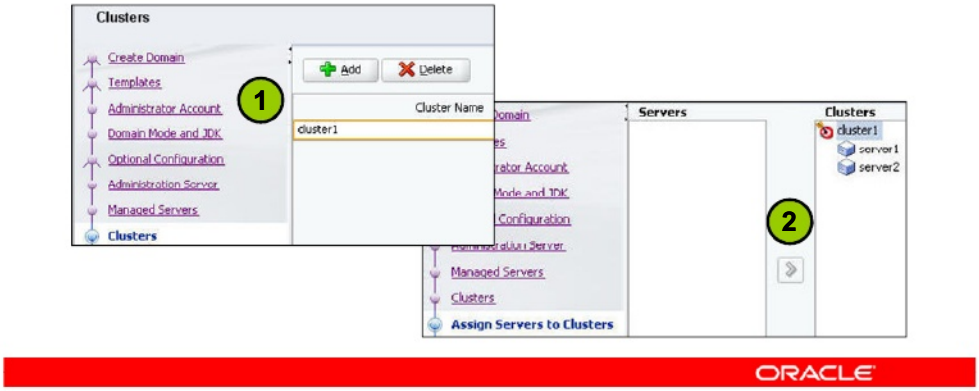
IP sockets provide a simple, high-performance mechanism for transferring messages and data between two applications. Clustered WebLogic Server instances use IP sockets for:

- Replicating HTTP session state and stateful session EJB state
- Accessing clustered objects that reside on a remote server instance (as in the multi-tier cluster architecture)

Creating a Cluster: Configuration Wizard

In the Configuration Wizard:

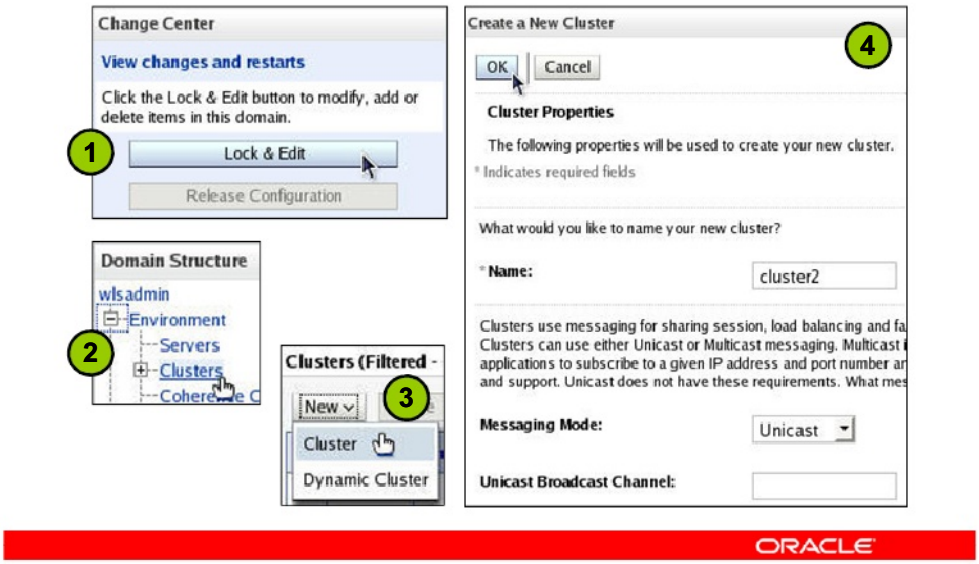
1. Add clusters.
2. Assign managed servers to them.



This is covered in the lesson titled “Creating Domains.”

Note that there is not a way to create dynamic clusters, server templates, or dynamic servers at the time of domain creation by using the Configuration Wizard. You can create a regular cluster by using the Configuration Wizard, and later create a server template and assign it to the cluster, which makes the cluster dynamic.

Creating a Cluster: Administration Console

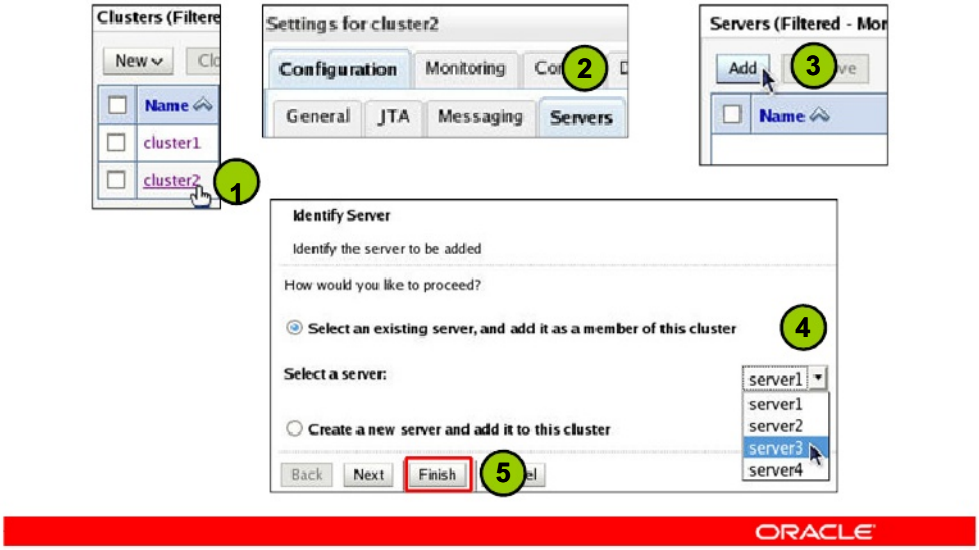


To create a new cluster, perform the following steps:

1. In the Change Center, click **Lock & Edit**.
2. Select **Clusters** under **Environment** in the Domain Structure.
3. Click the **New** button and select **Cluster**.
4. Give the cluster a unique name and select `Unicast` as the messaging mode. Click **OK**. If you have created a network channel for one-to-many cluster communication, enter it in the Unicast Broadcast Channel field. This field is optional. If left blank, each server's default network channel is used for this communication.

Note: The other messaging mode option is Multicast. If that is selected, you must also enter the Multicast Address and the Multicast Port.

Adding Servers to the Cluster: Administration Console



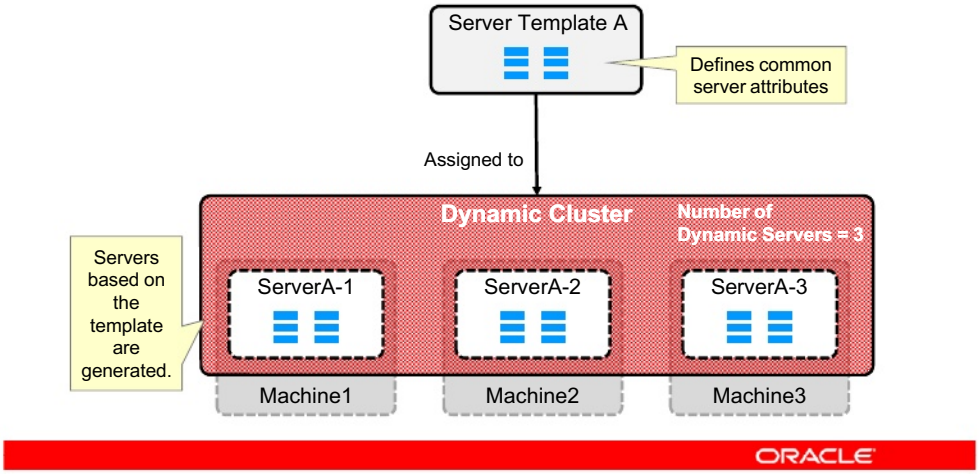
To add servers to the new cluster, perform the following steps (this assumes the configuration is still locked):

1. In the Clusters table, select the new cluster's name.
2. Select the **Configuration** tab and the **Servers** subtab.
3. Scroll down to the Servers table. Click the **Add** button.
4. To add an existing server to the cluster, choose **Select an existing server, and add it as a member of this cluster**. Then use the drop-down list to select a particular server. (The other option is to select **Create a new server and add it to this cluster**. You then click **Next** and are led through creating a new server.)
5. Click **Next** or **Finish**.
6. Repeat the process to add more servers to the cluster. (Not shown)
7. Finally, in the Change Center, click **Activate Changes**. (Not shown)

Note: You can also add a server to the cluster from the server's configuration. Lock the configuration. Select **Servers** under **Environment** in the Domain Structure. Click the name of a server in the Servers table. Select **Configuration** > **General**. Use the Cluster drop-down list to select a cluster. Click **Save**. Activate the changes. (The server cannot be running.)

Server Templates and Dynamic Clusters

A dynamic cluster is based on a server template.



Server Templates and Dynamic Clusters

- A server template defines server attributes.
  - Servers based on that template share those attributes.
    - If you change an attribute in the template, all of the servers based on that template change.
- A cluster can be associated with one server template. The cluster sets the number of dynamic servers needed.
  - That number of servers is generated and assigned to the cluster.
  - These servers show in the Servers table with the Type “Dynamic” (as opposed to “Configured”).
  - Attributes of dynamic servers that are server-specific are calculated when the servers are generated (for example, the server names).

Server templates allow you to define common attributes that are applied to a group of server instances. Because common attributes are contained in the template, you only need to change them in one place and they take effect in all of the servers that use the template. You use server templates in a heterogeneous server environment where server instances have a common set of attributes, but also have a set of unique, server-specific attributes. The primary use for server templates is in creating dynamic clusters.

Only one server template can be associated with a dynamic cluster. You use the server template to specify the configuration of the servers in the dynamic cluster, so that each server does not need to be manually created and configured for the cluster.

When configuring a dynamic cluster, you specify the number of server instances you anticipate needing at peak load. WebLogic Server creates the specified number of server instances and applies the calculated attribute values to each one. When you need additional capacity, you start one of the server instances not currently running, without having to first manually create and configure a new server and add it to the cluster.

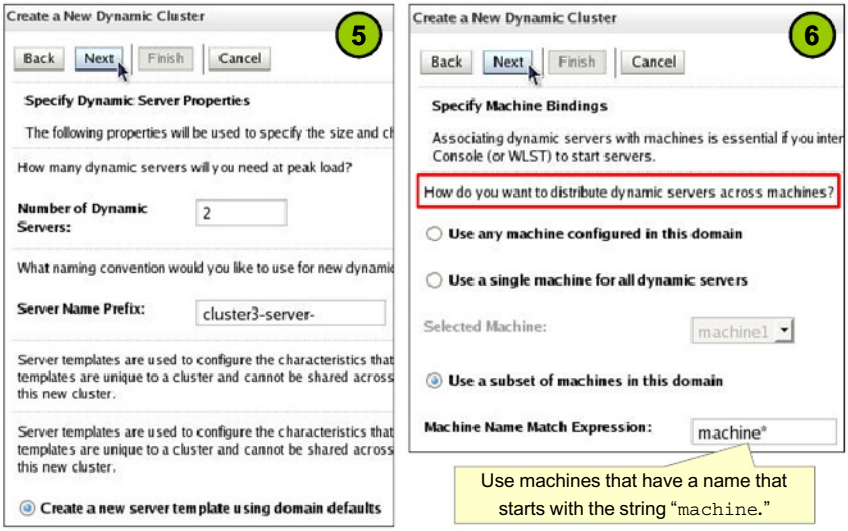
Creating a Dynamic Cluster

To create a new dynamic cluster, perform the following steps:

1. In the Change Center, click **Lock & Edit**.
2. Select **Clusters** under **Environment** in the Domain Structure.
3. Click the **New** button and select **Dynamic Cluster**.
4. Give the cluster a unique name and select `Unicast` as the messaging mode. Click **Next**. If you have created a network channel for one-to-many cluster communication, enter it in the Unicast Broadcast Channel field. This field is optional. If left blank, each server's default network channel is used for this communication.

   Note: The other messaging mode option is Multicast. If that is selected, you must also enter the Multicast Address and the Multicast Port.

Creating a Dynamic Cluster

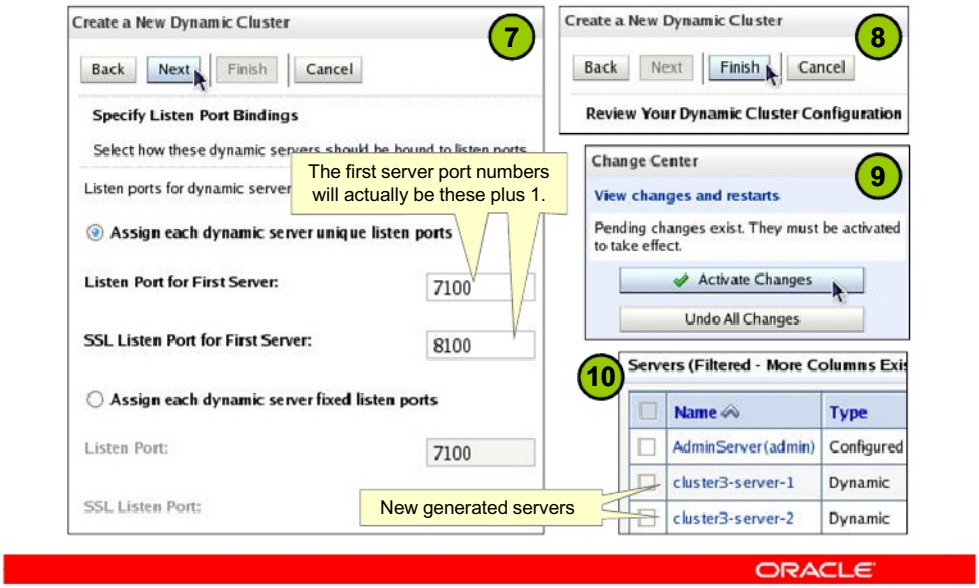


5. Enter the Number of Dynamic Servers. This is how many dynamic servers are created and placed in the cluster. This number should be the number of server instances you anticipate needing at peak load. Enter the Server Name Prefix. This is used to help generate the server names. Select **Create a new server template using domain defaults**. Click **Next**.

   Note: The other option for the server template is to select **Clone an existing server template for this cluster** and use the drop-down list to select an existing server template to copy.

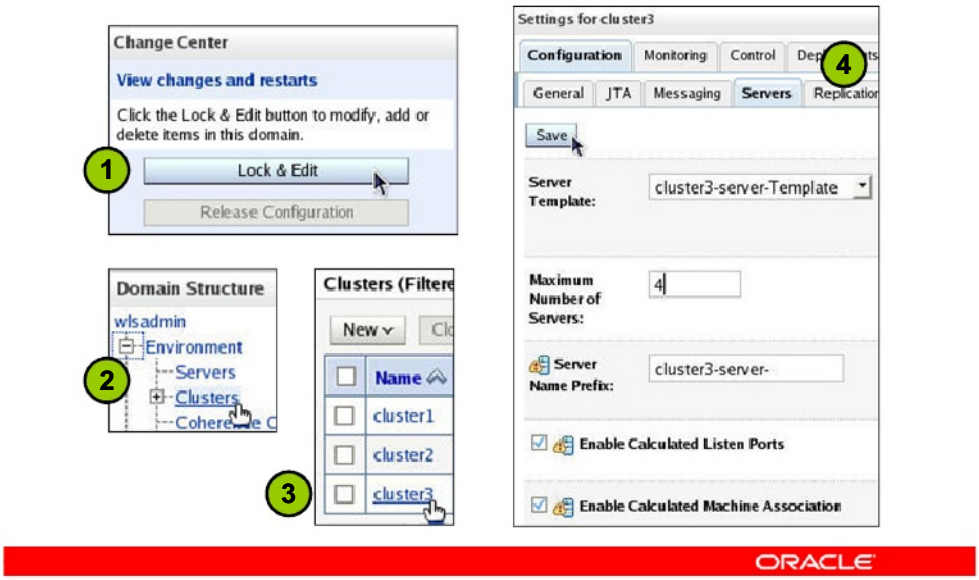
6. Select how to distribute the dynamic servers across machines, then click **Next**. The choices are:
   - **Use any machine configured in this domain:** The dynamic servers generated are assigned in a round-robin fashion to all machines defined in the domain.
   - **Use a single machine for all dynamic servers:** You want all dynamic servers on one machine. If this is chosen, you use the Selected Machine drop-down list to select that machine.
   - **Use a subset of machines in this domain:** The dynamic servers generated are assigned in a round-robin fashion to all machines defined in the domain that match the Machine Name Match Expression entered. An asterisk can be used as a wild card.

Creating a Dynamic Cluster



7. Select how to assign listen ports to the dynamic servers and then click **Next**. The two choices for how to do the port assignments are:
   - **Assign each dynamic server unique listen ports:** The dynamic servers generated are assigned unique listen ports by using the numbers entered in the Listen Port for First Server and SSL Listen Port for First Server fields. The first server port numbers are the values entered plus 1. Each subsequent server will have 1 added to the port numbers of the server generated before it.
   - **Assign each dynamic server fixed listen ports:** The dynamic servers generated are assigned the same listen ports entered in the Listen Port and SSL Listen Port fields. Note that this only works if the servers have different listen addresses. (No two servers can share the same listen address and listen port.)
8. Review the dynamic cluster and click **Finish**. The new server template and dynamic cluster are created.
9. In the Change Center, click **Activate Changes**.
10. After the changes are activated, new servers are generated, as shown in the Servers table.

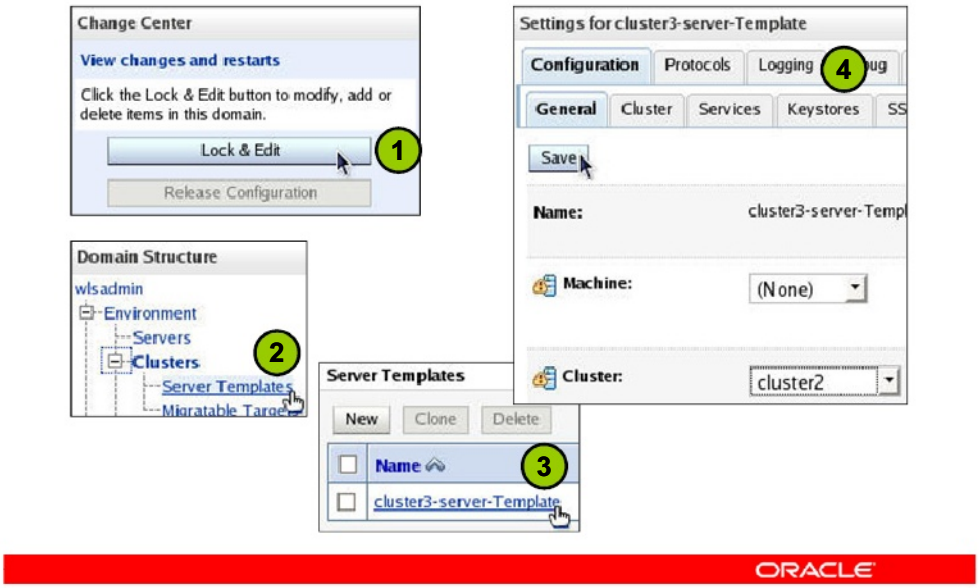
Editing the New Dynamic Cluster



To edit the new dynamic cluster, perform the following steps:

1. In the Change Center, click **Lock & Edit**.
2. Select **Clusters** under **Environment** in the Domain Structure.
3. Select the name of the new cluster in the Clusters table.
4. Select whichever tabs you want. Make changes to the cluster attributes. Click **Save**.
5. In the Change Center, click **Activate Changes**.

Editing the New Server Template



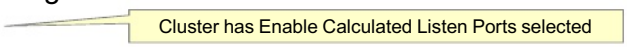
To edit the new server template, perform the following steps:

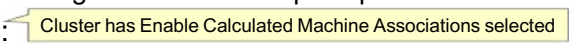
1. In the Change Center, click **Lock & Edit**.
2. In the Domain Structure, expand **Environment**, expand **Clusters**, and select **Server Templates**.
3. Select the name of the new server template in the Server Templates table.
4. Select whichever tabs you want. Make changes to the server template attributes. Click **Save**.
5. In the Change Center, click **Activate Changes**.

Dynamic Server Calculated Attributes

Dynamic servers are generated for a dynamic cluster based on the server template. Server-specific attributes are calculated:

- Server name: The Server Name Prefix followed by indexes in order, starting with `1`.
- Listen ports:
  - Dynamic: The port values entered in the template `+1` for the first server, `+2` for the second, and so on.
  - Static: Each server gets the same template port values
- Machine names:
  - No machine name match expression: All machines are rotated through as the servers are generated.
  - Machine name match expression: Only matching machines are rotated through as the servers are generated.





When you use a server template to create a dynamic cluster and specify the number of server instances you want, WebLogic Server uses calculated values for the following server-specific attributes:

- **Server name (always):** The calculated server name is controlled by the Server Name Prefix attribute. Server names are the specified prefix followed by an index. For example, if the prefix is set to `myserver-`, then the dynamic servers will have the names `myserver-1`, `myserver-2`, and so on.
- **Listen ports (optional—if the cluster has Enable Calculated Listen Ports selected):**
  - Dynamic:

    Standard port: The first server is Listen Port for First Server + 1, the second server is Listen Port for First Server + 2, and so on.

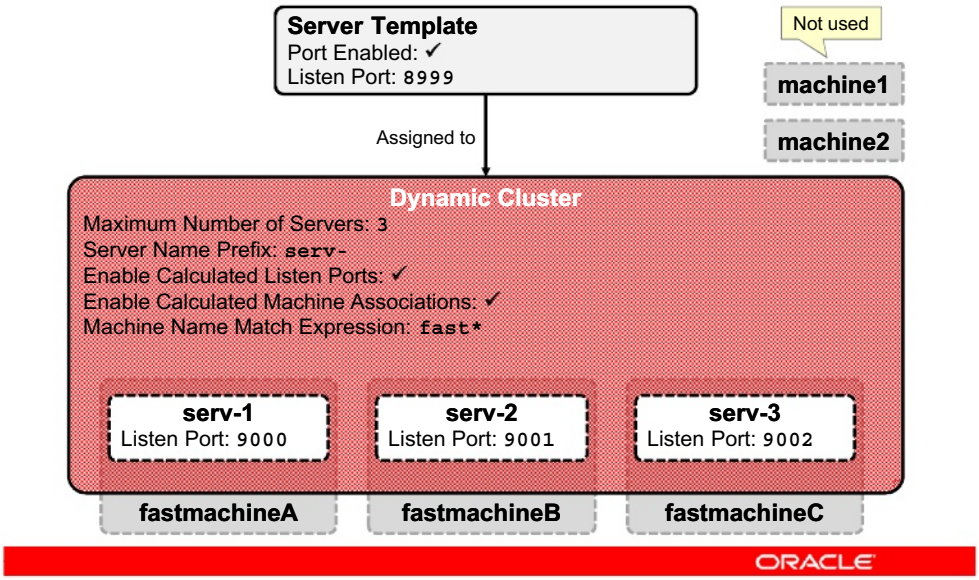
    SSL port: The first server is SSL Listen Port for First Server + 1, the second server is SSL Listen Port for First Server + 2, and so on.
  - Static:

    Standard port: All server listen ports equal the Listen Port value.

    SSL port: All server SSL ports equal the SSL Listen Port value.

- **Machines (optional—if the cluster has Enable Calculated Machine Associations selected):**
  - If no Machine Name Match Expression has been entered: All machines in the domain are used. Assignments are made in a round-robin fashion.
  - If a Machine Name Match Expression has been entered: Only those machines whose names match the expression are used. Assignments are made to matching machines in a round-robin fashion. Machine name match expressions can use asterisks for wildcards, and can list multiple expressions separated by commas. For example, say the domain has these machines defined: `mach1`, `mach2`, `mach3`, `fastmach1`, `fastmach2`, `fastmach3`, and `fastmach4`. And the Machine Name Match Expression is set to: `mach1,fast*`. The machines would be assigned in this order: `mach1`, `fastmach1`, `fastmach2`, `fastmach3`, `fastmach4`, `mach1`, `fastmach1`, and so on.
- **Network Channel (Access Point) listen ports (optional—if the server template has a Network Channel defined within it):** The first server is the network channel port + 1, the second server is the network channel port + 2, and so on.

Dynamic Server Calculated Attributes: Example



The three servers generated for the dynamic cluster have a name that starts with `serv-` and ends in an index, starting with `1`. The listen ports for the servers start with the template listen port `+1`. The servers are assigned to machines already defined, but only those machines with names that start with "`fast`."

Comparing Configured and Dynamic Clusters

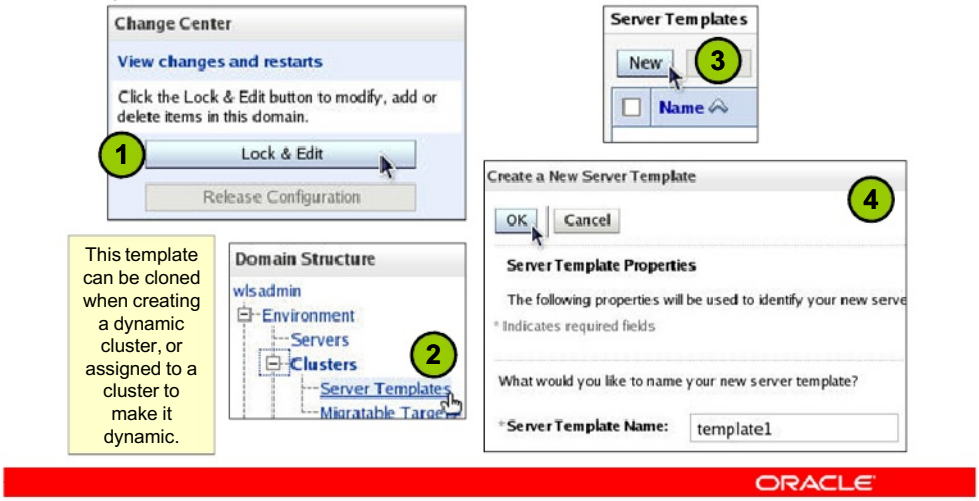
Feature	Configured Cluster	Dynamic Cluster
Create with the Admin Console / WLST	Yes	Yes
Create with the Configuration Wizard	Yes	No
Edit individual server attributes	Yes	No
Servers generated automatically	No	Yes
Can contain configured servers	Yes	Yes
Can contain dynamic servers	No	Yes
Supports service-level migration	Yes	No
Supports whole-server migration	Yes	No

ORACLE

A configured cluster that is assigned a server template and has dynamic servers generated for it becomes a dynamic cluster. A dynamic cluster can contain both configured and dynamic servers.

Creating a Server Template

You can create a server template independently from creating a dynamic cluster:



To create a new server template, perform the following steps:

1. In the Change Center, click **Lock & Edit**.
2. In the Domain Structure, expand **Environment**, expand **Clusters**, and select **Server Templates**.
3. Click the **New** button.
4. Give the server template a unique name and click **OK**.

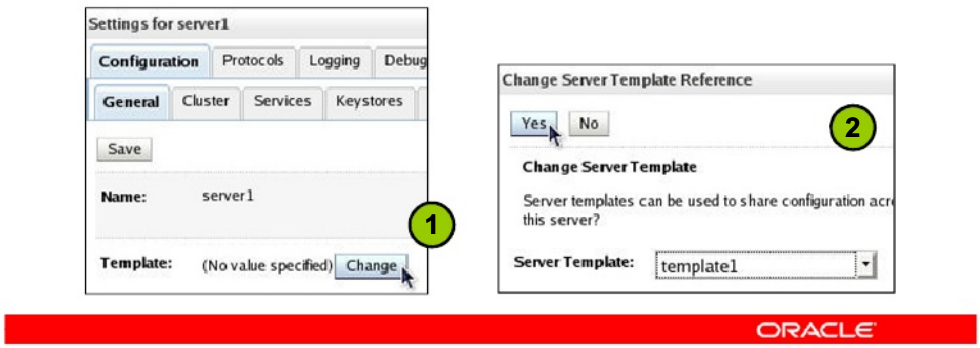
Creating a Server Template

5. In the Server Templates table, select the name of the new template.
6. Select **Configuration** > **General**.
7. Enter any values for attributes that you want shared among servers created from this template. Some attribute values can be overridden when the template is assigned to a cluster. Other attribute values are used in calculations. For example, when a value for Listen Port is entered, it is used as the starting point for listen ports for the servers generated from the template. After entering the values, click **Save**.
8. Select any other tabs and enter whatever attribute values you want shared among these servers. Remember to click **Save** after entering values on a page.
9. In the Change Center, click **Activate Changes**.

Server Templates and Configured Servers

In addition to using server templates to define the servers in a dynamic cluster, a server template can be assigned to any number of configured servers, so those servers can share common, nondefault attributes.

- The attributes can be overridden by the individual servers.



1. After creating the server template and locking the configuration, in the Servers table, select the server. Ensure that **Configuration** > **General** is selected. Click the **Change** button for the Template field.
2. Select the server template from the drop-down list and click **Yes**. Then activate the changes.

Quiz

The multi-tier cluster architecture allows you to load balance EJB calls. But, the basic (single-tier) architecture has an EJB-related advantage over multi-tier. The advantage is:

a. It cannot use EJBs, which makes development simpler

b. This is a trick question, because the single-tier architecture has no EJB-related advantages

c. All EJB calls are local and, therefore, faster

Answer: c

Quiz

A dynamic cluster is based on:

a. One server template
b. Multiple server templates
c. A cluster proxy
d. A domain template

Answer: a

Summary

In this lesson, you should have learned how to:

- Describe two cluster architectures: basic and multi-tier
- Create and configure a cluster
- Create and configure a dynamic cluster

Practice 12-1 Overview: Configuring a Cluster

This practice covers creating a cluster by using the administration console.

Practice 12-2 Overview: Configuring a Dynamic Cluster

This practice covers creating a dynamic cluster by using the administration console.

13 Clusters

Proxies and Sessions

Objectives

After completing this lesson, you should be able to:

- Install Oracle HTTP Server
- Configure Oracle HTTP Server as a cluster proxy
- Configure session failover
- Configure replication groups

A Cluster Proxy for a Web Application Cluster

- A cluster proxy provides load balancing and failover for a web application cluster. It gives the cluster its “single server” appearance.
- There are basically two kinds of cluster proxies:
  - A web server with the WebLogic proxy plug-in
  - A hardware load balancer

Cluster Proxy	Advantages	Disadvantages
Web server with plug-in	• Low cost (or free) • You probably already have experience with the web server	• Only round-robin load balancing available • Must configure the plug-in
Hardware load balancer	• More sophisticated load balancing algorithms • No plug-in configuration	• Cost • Must be compatible with the WebLogic Server session cookie

ORACLE

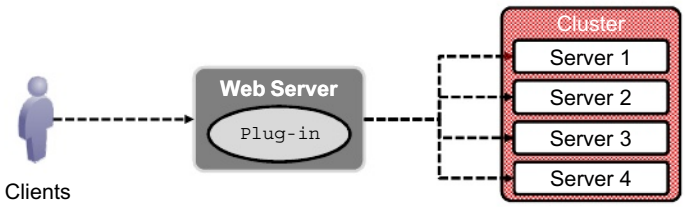
Cluster Proxies are how clients interact with a web application cluster, whether they are hardware or software based. You have two basic choices of cluster proxy: a web server using a plug-in or a hardware load balancer (such as F5 BIG-IP).

Hardware load balancers must support a compatible passive or active cookie persistence mechanism. Passive cookie persistence enables WebLogic Server to write a cookie containing session information through the load balancer to the client. You can use certain active cookie persistence mechanisms with WebLogic Server clusters, provided the load balancer does not modify the WebLogic Server 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.

A WebLogic Server proxy plug-in is available for Netscape Enterprise Server, Apache HTTP Server, Microsoft Internet Information Server (IIS), and Oracle HTTP Server (which is based on Apache). These plug-ins provide round-robin load balancing to the servers in the cluster. They use the WebLogic Server session cookie information to route requests to the server that has a client’s session data.

Proxy Plug-Ins

- A proxy plug-in:
  - Load balances client requests to clustered WebLogic Server instances in a round-robin fashion
  - Avoids routing requests to failed servers in the cluster
  - Routes requests based on WebLogic Server session cookie information



The web server with the proxy plug-in may do more than just load balance requests to WebLogic Server instances in a cluster. In some architectures, the web server serves up static files (HTML files and images, for example) and only passes through requests to WebLogic Server for “dynamic” pages (JSPs or calls to Servlets). To the web browser, the HTTP responses appear to come from one source—the web server. A variation of that architecture is to use a hardware load balancer in front of a bank of web servers (serving up static content), which are in front of a cluster of WebLogic Server instances (returning “dynamic” content).

Oracle WebLogic Server plug-ins can provide efficient performance by reusing connections from the plug-in to WebLogic Server (“keep-alive” connections).

Oracle HTTP Server (OHS)

OHS is a web server that is:

- A component of the Oracle Web Tier Suite
- Based on Apache HTTP Server
- Installed with the WebLogic Server plug-in module (`mod_wl_ohs`) by default
  - The plug-in must be configured by using the `mod_wl_ohs.conf` file.
- Managed and monitored by using the Oracle Process Manager and Notification Server (OPMN)
  - OPMN manages and monitors non-Java components of Oracle Fusion Middleware.
  - OPMN can be accessed by using the `opmnctl` command-line utility.

OHS is based on the Apache web server. OHS supports single sign-on, clustered deployment and high availability, and Web Cache.

Configuration of Oracle HTTP Server is specified through directives in configuration files in the same manner as with Apache HTTP Server.

A `mod_wl_ohs` module is available in OHS. This module enables you to integrate your WebLogic Server environment with OHS immediately after the configuration of the OHS instance and the domains.

OHS directories are divided between the Oracle home and the Oracle instance. The Oracle home directories are read-only, and contain the Oracle Fusion Middleware binaries. The Oracle instance directories contain the modules, applications, and logs for OHS. Each OHS component has a root configuration directory found at `<instance>/config/OHS/<component>`, which includes the WLS plug-in configuration file, `mod_wl_ohs.conf`. Similarly, each component's log files are found at `<instance>/diagnostics/logs/OHS/<component>`.

OPMN provides Oracle Fusion Middleware system (non-Java) components with process management and failure detection. It consists of the Oracle Process Manager (PM) and the Oracle Notification Server (ONS). PM is responsible for starting, restarting, stopping, and monitoring the system processes. ONS is the transport mechanism for failure, recovery, startup, and other related notifications between components in Oracle Fusion Middleware.

Installing and Configuring OHS (Part of Oracle Web Tier): Overview

1. Download and unzip the Web Tier installer.
2. Run the executable under `Disk1`: `runInstaller`.
   A. Choose the **Install Software - Do Not Configure** option.
   B. Specify the Web Tier installation location.
3. Configure an OHS instance by navigating to `<WEB_TIER>/bin` and running the Web Tier Configuration Wizard: `config.sh`.
   A. Under Configure Components, select **Oracle HTTP Server**.
   B. Enter the Instance Home Location, the Instance Name, and the OHS Component Name. This location is called "oracle instance."
   C. Configure the ports (select either **Auto Port Configuration** or a port configuration file).
   D. Click **Configure**, and at 100% complete, click **Finish**.

An "oracle instance" location contains one or more Fusion Middleware system components, such as Oracle Web Cache or Oracle HTTP Server. The oracle instance directory contains updatable files, such as configuration files, logs files, and temporary files.

A Web Tier port configuration file is a text file that specifies ports for web tier components. There is a sample file, `staticports.ini`, under the unzipped installation directories here: `Disk1/stage/Response`. You can copy that file and modify it to set the ports to the values you want. Here is a sample:

```
[OPMN]
OPMN Local Port = 6700
[OHS]
OHS Port = 7777
OHS Proxy Port = 7779
OHS SSL Port = 7778
```

Configuring OHS as the Cluster Proxy

Modules extend the functionality of OHS to enable it to integrate with other Fusion Middleware components.

- The proxy plug-in for WebLogic Server is called `mod_wl_ohs.so` and is found here: `<WEB_TIER>/ohs/modules`.
  - The plug-in is already installed, but must be configured.
- Configuration files for OHS are found here: `<ORACLE_INSTANCE>/config/OHS/`*`OHS_instance_name`*.
  - The main configuration file is `httpd.conf`. It contains an include directive for the WebLogic plug-in configuration file:
  - `mod_wl_ohs.conf`. This is the file you edit to configure OHS to proxy a WebLogic Server cluster.

Modules extend the basic functionality of OHS, and support integration between OHS and other Oracle Fusion Middleware components. The `mod_wl_ohs.so` module is installed and loaded out-of-the-box with Oracle HTTP Server, but it is not configured by default. Therefore, you must configure it to specify the application requests that the module should handle. The `mod_wl_ohs` module enables requests to be proxied from an OHS to Oracle WebLogic Server. The configuration for this module is stored in the `mod_wl_ohs.conf` file, which can be edited manually with a text editor.

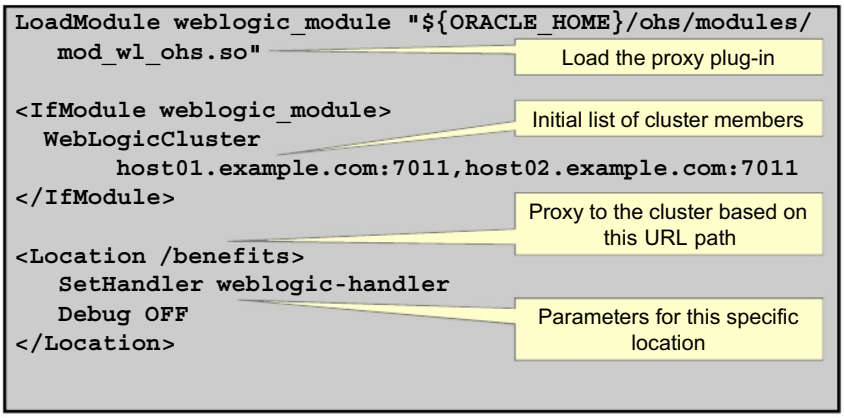
`httpd.conf` and `mod_wl_ohs.conf`

- The include directive in `httpd.conf` looks like this (all on one line):

```
include "${ORACLE_INSTANCE}/config/${COMPONENT_TYPE}/
                 ${COMPONENT_NAME}/mod_wl_ohs.conf"
```

- The `mod_wl_ohs.conf` file has various directives, but the `WebLogicCluster` directive is the most important.
  - It specifies the initial list of servers in the cluster, giving their host names and ports.
    - Remember, you do not need to update this list in the configuration file to add or remove servers from the cluster. This is the *initial* list of cluster members. Once the cluster is running, the plug-in uses the dynamic server list.

mod_wl_ohs.conf



To proxy requests to a single server, use the `WebLogicHost` and `WebLogicPort` parameters. To proxy requests to a cluster of WebLogic Servers, use the `WebLogicCluster` parameter instead.

To proxy requests by path, use the `Location` block and the `SetHandler` statement. `SetHandler` specifies the handler for the plug-in module, and should be set to `weblogic-handler`. To proxy requests by MIME type, add a `MatchExpression` line to the `IfModule` block. Note that if both MIME type and proxying by path are enabled, proxying by path takes precedence over proxying by MIME type. You can also use multiple `MatchExpressions` lines.

Some Plug-in Parameters

Parameter	Description
`WebLogicHost, WebLogicPort`	Proxy to a single server with this host and port
`WebLogicCluster`	Proxy to this initial list of clustered servers
`MatchExpression`	Proxy requests for files of this MIME type
`PathTrim`	Remove this text from the incoming URL path before forwarding a request.
`PathPrepend`	Add this text to the incoming URL path before forwarding a request.
`ErrorPage`	URL to direct users to if all servers are unavailable
`WLExcludePathOrMimeType`	Do not proxy for this specific URL path or MIME type.
`WLProxySSL`	Set to `ON` to establish an SSL connection to WebLogic if the incoming request also uses HTTPS.
`MaxPostSize`	Maximum allowable size of POST data, in bytes
`Debug`	Sets the type of logging performed

ORACLE

- **`WebLogicHost,WebLogicPort`**: WebLogic Server host (or virtual host name as defined in WebLogic Server) to which HTTP requests should be forwarded. Port at which the WebLogic Server host is listening for connection requests from the plug-in.
- **`WebLogicCluster`**: Comma-separated list of `host:port` for each of the initial WebLogic Server instances in the cluster. The server list specified in this property is a starting point for the dynamic server list that the servers and plug-in maintain. WebLogic Server and the plug-in work together to update the server list automatically with new, failed, and recovered cluster members.
- **`MatchExpression`**: When proxying by MIME type, set the filename pattern inside of an `IfModule` block using the `MatchExpression` parameter.
- **`PathTrim`**: Specifies the string trimmed by the plug-in in the `{PATH}/{FILENAME}` portion of the srcinal URL, before the request is forwarded to WebLogic Server.
- **`ErrorPage`**: You can create your own local error page that is displayed when your web server is unable to forward requests to WebLogic Server.
- **`WLExcludedPathOrMimeType:`** This parameter allows you to exclude certain requests from proxying.

- **`WLProxySSL`**: Set this to `ON` to establish an SSL connection to WebLogic Server if the incoming request uses HTTPS.
- **`MaxPostSize`**: Maximum allowable size of POST data, in bytes. If the content-length exceeds this value, the plug-in returns an error message. If set to `-1`, the size of POST data is not checked. This is useful for preventing denial-of-service attacks that attempt to overload the server.
- **`Debug`**: Sets the type of logging performed for debugging operations. The debugging information is written to the `/tmp/wlproxy.log` file. Some of the possible values for this parameter are:
  - **`ON`**: The plug-in logs informational and error messages.
  - **`OFF`**: No debugging information is logged.
  - **`ERR`**: Only error messages are logged.
  - **`ALL`**: The plug-in logs headers sent to and from the client, headers sent to and from WebLogic Server, information messages, and error messages.

Starting and Stopping OHS

- OHS is managed by OPMN.
  - The command-line interface to OPMN is `opmnctl`.
- `opmnctl` examples:

Start OPMN and all managed processes, if not already started:
```
$> ./opmnctl startall
```

Start all OHS processes, if not already started:
```
$> ./opmnctl startproc process-type=OHS
```

Get the name, status, memory usage, and port number of processes:
```
$> ./opmnctl status -l
```

Stop all OHS processes:
```
$> ./opmnctl stopproc process-type=OHS
```

Oracle HTTP Server is managed by OPMN. You can use `opmnctl` to start, stop, and restart OHS.

You can include the path to the `opmnctl` location (`<ORACLE_INSTANCE>/bin`) or change to the `opmnctl` directory before using the `opmnctl` commands. `<ORACLE_INSTANCE>` is the location where this OHS instance has been configured.

The available `opmnctl` commands include:

- **`start`**: Start the OPMN server for a local Oracle instance without starting system processes.
- **`startall`**: Start OPMN as well as the system processes for a local Oracle instance. **`startall`** is equivalent to `start` followed by **`startproc`** without arguments.
- **`stopall`**: Shut down the OPMN server as well as the system processes for the local Oracle instance. This request operates synchronously; it waits for the operation to complete before returning.
- **`startproc`**, **`restartproc`**, **`stopproc`**: Use these commands to start, restart, or stop system processes. The OPMN server must be up and running.

The following attributes are supported. Any of these attributes may be omitted, and treated as a wild card:

- **`ias-component`**: The name of a specific managed process, as defined in `opmn.xml`.
- **`process-type`**: The type of managed process to command, as defined in `opmn.xml`.
- **`process-set`**: The name of a custom process group defined in `opmn.xml`.

Verifying that OHS Is Running

1. View the port on which OHS is running by using the `opmnctl status -l` command.

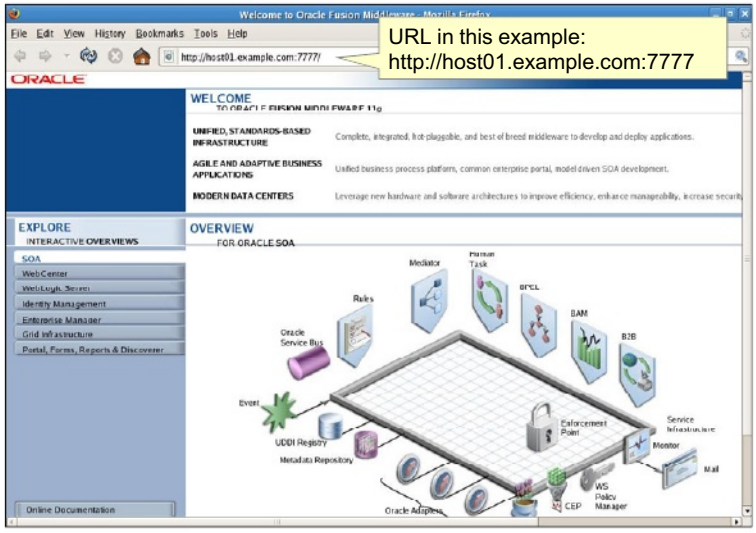
```
$> ./opmnctl status -l

Processes in Instance: webtier

---------+---------+-----+------+-...+--------
ias-     |process- |     |      |    |
component|type     | pid |status|    | ports
---------+---------+-----+------+-...+--------
ohs1     | OHS     | 2598|Alive |    | https:7779,
                                     | https:7778,
                                     | http:7777
```

2. In a web browser, enter the URL of the host name where OHS was started followed by the discovered HTTP port.

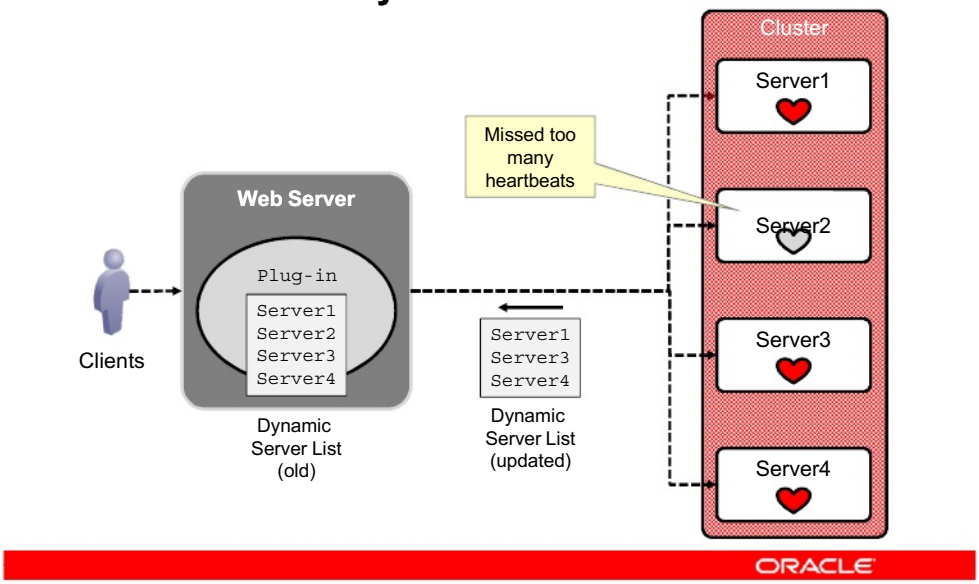
You can verify that you are able to access the applications deployed to a cluster through OHS by directing your request to the port on which OHS is listening for requests. You can discover the HTTP Listen port of OHS by using the `opmnctl status -l` command. In the slide, OHS is running (HTTP) on port `7777`.

Enter the host and port in a web browser. If OHS is running, you will see a splash page, as shown in the next slide.

Successful Access of OHS Splash Page



Failover: Detecting Failures and the Dynamic Server List



The cluster and the proxy maintain a dynamic server list that has within it all viable servers in the cluster. When a server in the cluster fails it is detected and the list is updated. If a server misses too many heartbeats, it is marked as failed. Also, if a socket to a server in the cluster (from another cluster server) closes unexpectedly, the server is marked as failed.

The dynamic server list is also updated when new servers in the cluster are started.

Failover: Detecting Failures and the Dynamic Server List

- A clustered server detects the failure of another server in the cluster when:
  - A socket to that server unexpectedly closes
  - That server misses three* heartbeats
- In either case, that server is marked as "failed."
- Responses from a clustered server to a cluster proxy include the "dynamic server list," a list of all the current, viable servers in the cluster.
  - The list lets the proxy know which servers it can use.
  - The list is updated not only when servers fail, but also when new servers are added to the cluster.

* This number is configurable.

When you specify the list of WebLogic Server instances for a cluster proxy, the plug-in uses that list as a starting point for load balancing among the members of the cluster. After the first request is routed to one of these servers, a dynamic server list is returned in the response that contains an updated list of servers in the cluster. The updated list adds any new servers in the cluster and deletes any that are no longer part of the cluster or that have failed. The dynamic server list is returned with each clustered server response so that the proxy always has an up-to-date list of viable servers.

To configure how many heartbeats can be missed: In the admin console, select the cluster, then select **Configuration** > **Messaging**. Under **Advanced**, change **Idle Periods Until Timeout**. The default is `3`.

HTTP Session Failover

- Web applications store objects in HTTP sessions to track information for each client in memory.
- When an instance of WebLogic Server creates a session, it writes a cookie to the client's web browser indicating that it is the server for this client. Subsequent requests from that client are routed by the proxy to this server.
- If the server fails, its clients must be routed to other servers in the cluster, and session information is lost.
- WebLogic Server supports several strategies so that the session information is not lost when a server fails:
  - In-memory session replication — Recommended, as it is the fastest
  - JDBC (database) session persistence
  - File session persistence

Java web application components, such as Servlets and JavaServer Pages (JSPs), can maintain data on behalf of clients by storing objects in the `HttpSession`. An `HttpSession` is available on a per-client basis. Once an instance of WebLogic Server creates a session for a client, it also writes a cookie to the client's web browser. This cookie indicates which server has this client's session. The cluster proxy checks this cookie on subsequent client requests, and routes the client to the instance of WebLogic Server that has the client's session.

If the server that has the client's session fails, when the client makes their next request, they cannot be routed to that server. The proxy chooses another server. That server does not have the client's session, so information about the client is lost.

To provide transparent failover for web applications, replication or shared access to the information in each `HttpSession` object must be provided. This is accomplished within WebLogic Server by using in-memory replication, file system persistence, or database persistence. A web application chooses which session failover option to use in the WebLogic Server deployment descriptor, `weblogic.xml`. Each option has its own configurable parameters that are also entered in `weblogic.xml`.

Note that in-memory replication has two options, synchronous and asynchronous. The asynchronous option replicates data in batches to improve cluster performance.

Configuring Web Application Session Failover: `weblogic.xml`

- Developers configure sessions in `weblogic.xml`, under the `<session-descriptor>` tag.
- Its subtag, `<persistent-store-type>`, configures session failover:

`<persistent-store-type>`	Description
`memory`	No session replication or persistence
`replicated`	In-memory session replication
`replicated_if_clustered`	The same as `memory` if deployed to stand-alone servers, the same as `replicated` if deployed to a cluster

The value of the `<persistent-store-type>` tag determines session failover:

- **`memory`**: No session replication or persistence
- **`replicated`**: In-memory session replication. The syncing of sessions between the primary and secondary servers occurs synchronously. Note that it is an error to deploy a web application with this option to stand-alone servers.
- **`replicated_if_clustered`**: If the web application is deployed to stand-alone servers, this option is the same as `memory`. If the web application is deployed to a cluster, this is the same as `replicated`.

Configuring Web Application Session Failover: `weblogic.xml`

<persistent-store-type>	Description
async_replicated	In-memory session replication with syncing done in batches
async_replicated_if_clustered	The same as memory if deployed to stand-alone servers, the same as async-replicated if deployed to a cluster
file	File-based persistence of sessions
jdbc	Database persistence of sessions
async_jdbc	Database persistence of sessions with updates done in batch
cookie	All session data stored in cookies

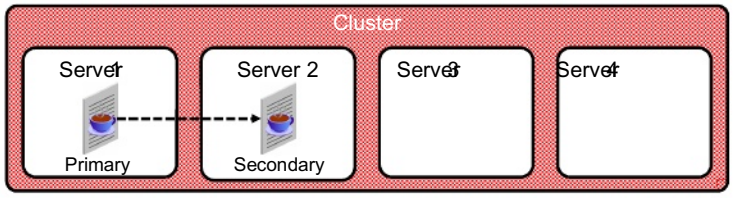
The value of the `<persistent-store-type>` tag determines session failover:

- **`async_replicated`**: In-memory session replication, but the syncing of sessions between the primary and secondary servers is done in batches, rather than synchronously. Note that it is an error to deploy a web application with this option to stand-alone servers.
- **`async_replicated_if_clustered`**: If the web application is deployed to stand-alone servers, this option is the same as `memory`. If the web application is deployed to a cluster, this is the same as `async_replicated`.
- **`file`**: File-based session persistence. This requires another subtag, `<persistent-store-dir>`, which specifies the directory where files containing session data are placed. This directory must be accessible to all servers in the cluster.

- **jdbc**: Session data is stored in a database. This requires another subtag, <persistent-store-pool>, which specifies the data source used to access the database. This data source must be deployed to all servers in the cluster. The database that is accessed through this data source must have a table named WL_SERVLET_SESSIONS with certain columns of particular data types. For more information, see the chapter titled "Using Sessions and Session Persistence" in the *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server* document.
- **async_jdbc**: The same as the jdbc option, except the syncing of sessions to the database is done in batches, rather than synchronously.
- **cookie**: All session data is stored in cookies in the client's web browser. If this option is chosen, only string data can be stored in the session.

In-Memory Session Replication

- Each client's session object exists on two servers:
  - Primary
  - Secondary
- The WebLogic Server session cookie stores both the client's primary and secondary servers.
- Each update to the primary session object is automatically replicated to the secondary server, either synchronously (the default) or asynchronously (batch).

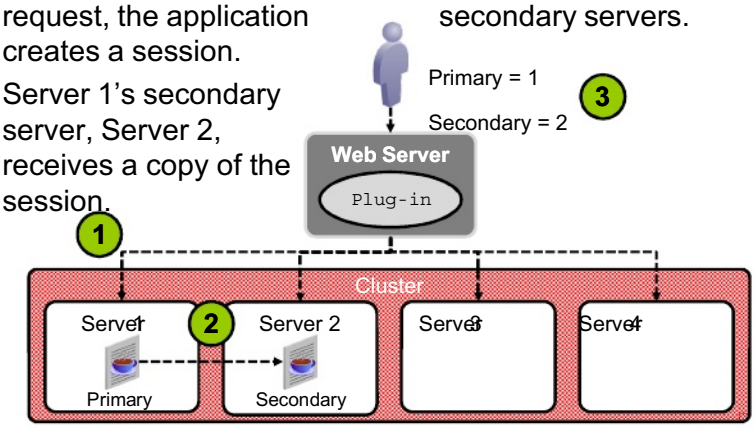


Using in-memory replication, WebLogic Server copies session state from one server instance to another. The primary server stores the primary session state (the primary server is the server to which the client is connected when a session is first created). A replica of the session state is stored on another instance of WebLogic Server in the cluster (the secondary server). The replica is kept up-to-date so that the data can be used if the primary server fails.

The default session replication is synchronous. The asynchronous option replicates data in batches to improve cluster performance.

In-Memory Replication: Example

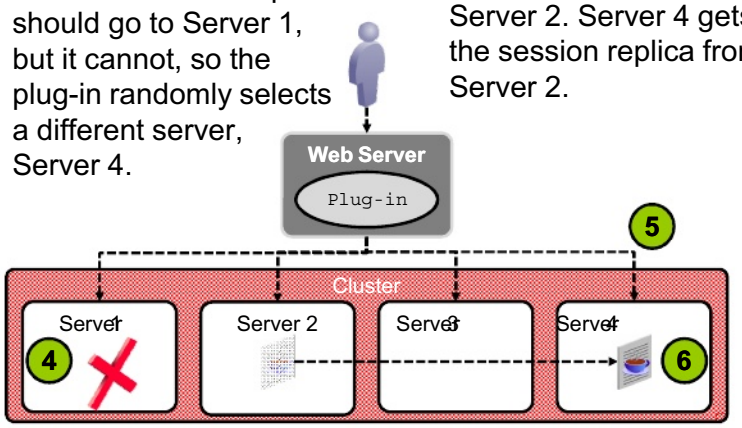
1. A client is load balanced to Server 1. On this request, the application creates a session.
2. Server 1's secondary server, Server 2, receives a copy of the session.
3. The cookie is written to track the primary and secondary servers.



1. When a client first accesses the web application through the proxy plug-in, the plug-in load balances the request to one of the servers in the cluster (in the example, Server 1). The web application running on Server 1, because the application wishes to remember something about the client, creates a session for the client and stores something in it.
2. Server 1 is this client's primary server. To provide failover services for the web application, the primary server replicates the client's session state to a secondary 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, Server 2 is the secondary server for Server 1, and gets a replica of the client's session object.
3. When WebLogic Server responds to the client, it writes a cookie to the client's browser. This cookie contains which server is the primary (Server 1, in the example) and which is the secondary (Server 2, in the example). The cookie also contains the client's session ID. Subsequent requests from this client are routed (if possible) to the primary server, Server 1, which has the client's session information.

In-Memory Replication: Example

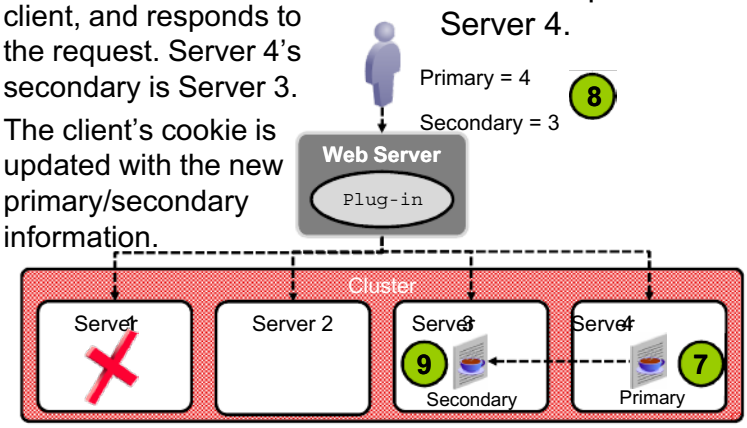
4. Server 1 fails.
5. The client's next request should go to Server 1, but it cannot, so the plug-in randomly selects a different server, Server 4.
6. The client's cookie stores the secondary server, Server 2. Server 4 gets the session replica from Server 2.



4. Server 1 fails.
5. When a subsequent request from the Server 1 client comes in, the proxy must use another server. It picks that server at random. In this example, Server 4 is chosen.
6. The proxy uses the client's cookie information to determine the location of the secondary server that holds the session replicas of the failed server. In this example, the secondary server is Server 2. The new primary server, Server 4, contacts the old secondary server, Server 2, and retrieves the replicated session object from it.

In-Memory Replication: Example

7. Server 4 is now the primary server for the client, and responds to the request. Server 4's secondary is Server 3.
8. The client's cookie is updated with the new primary/secondary information.
9. Server 3 stores the session replica of Server 4.



7. Server 4 responds to the request. The session information for the client is available to Server 4, so the client does not realize that a different server is responding. Server 4 is the new primary server for this client.
8. Server 4 has a secondary server, in this example, Server 3. Part of Server 4's response is to update the client's cookie information. Server 4 is now the primary, with Server 3 as the secondary.
9. Server 3, as the secondary server, stores a replica of this client's session object for the new primary server, Server 4.

Configuring In-Memory Replication

- Configure in-memory replication in the `weblogic.xml` deployment descriptor.

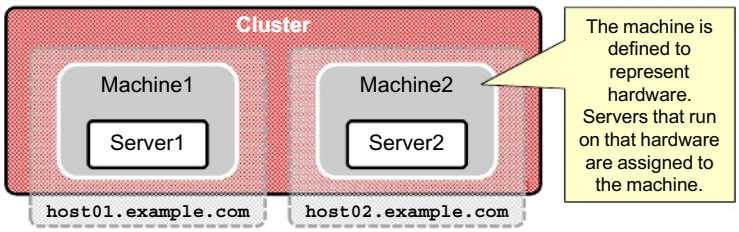
```
...
<session-descriptor>
   <persistent-store-type>replicated_if_clustered
   </persistent-store-type>
</session-descriptor>
...
```

weblogic.xml

In the `weblogic.xml` deployment descriptor file, set the `persistent-store-type` parameter in the `session-descriptor` element to `replicated`, `replicated_if_clustered`, `async_replicated`, or `async_replicated_if_clustered`.

Machines

- WebLogic Server uses machine definitions and the servers assigned to them to indicate which managed servers run on what hardware.
- WebLogic Server takes machine definitions into account when it chooses a secondary server as a backup for session information.
  - It prefers one on a different machine than the primary server.

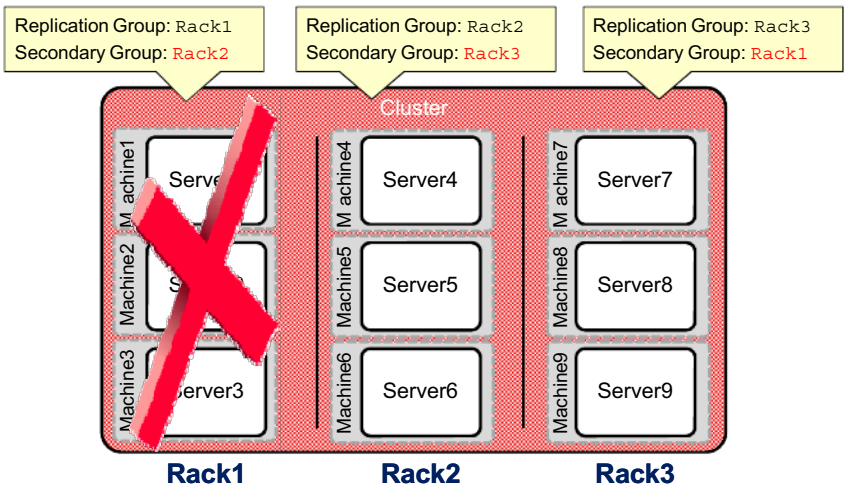


Secondary Server and Replication Groups

- A replication group is a logical grouping of servers in a cluster.
- WebLogic Server allows you to influence how secondary servers are chosen by configuring replication groups and configuring a server's "preferred secondary group."
- When choosing a secondary server, WebLogic Server attempts to:
  - Choose one in the primary server's preferred secondary group, if it is configured
  - Choose a server on a different machine
  - Avoid choosing a server in the same replication group

In addition to taking into account machine definitions, WebLogic Server allows you to further control how secondary servers are chosen by using replication groups. A replication group is a preferred list of clustered instances to use for storing session replicas. When you configure a server instance that participates in a cluster, you can assign the server instance membership in a replication group. You can also assign a preferred secondary replication group to be considered for replicas of the session states that reside on the server.
When a web client attaches to a cluster and a session is created, the WebLogic Server instance that is now the primary server ranks other servers in the cluster to determine which server should be the secondary. Server ranks are assigned using the server's machine and participation in a replication group.

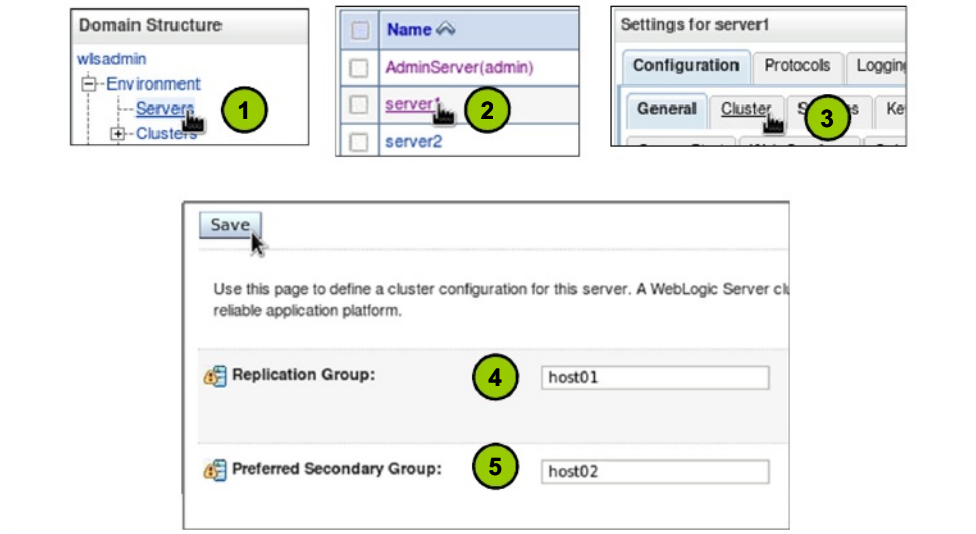
Replication Groups: Example



This slide shows an example of using replication groups to persuade WebLogic Server into placing the primary sessions on servers running on a group of machines that run on the same physical rack. The cluster spans multiple machines that run on three different physical racks. In this example, all servers are configured with a replication group name that matches the rack name where they are running. So servers running on `Rack1` have a replication group setting of `Rack1`. All servers are configured with a secondary replication group name that matches a rack name that is on a different rack. The configured secondary group for servers running on `Rack1` is `Rack2`. This means that primary sessions in-memory on servers running on `Rack1` have their secondary sessions replicated to one of the servers running on `Rack2`. Each server in this cluster is configured in this way to ensure that the primary session is always on a server within one rack while the secondary is located on a server in another rack.

If somehow `Rack1` becomes totally unavailable, client requests will fail over to other servers in the cluster and are guaranteed to recover their session state because the replication group configuration ensured that secondary sessions were located on another rack.

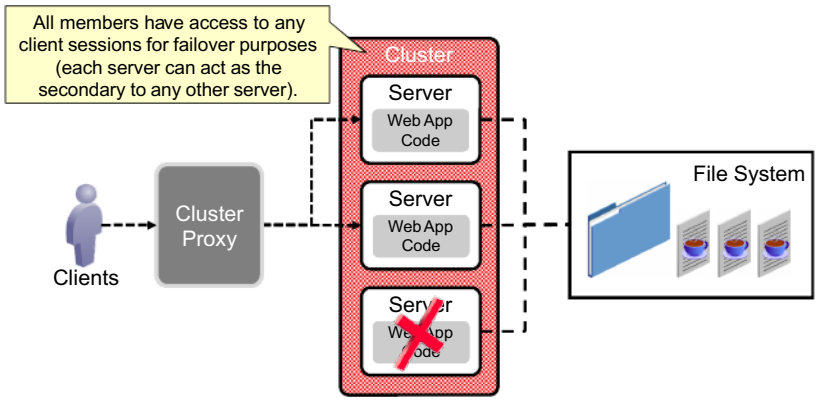
Configuring Replication Groups



1. In the Domain Structure, expand **Environment** and select **Servers**.
2. Select the server for which you want to configure a replication group.
3. Select the **Configuration** > **Cluster** tabs.
4. Enter the name of the replication group that this server belongs to and the preferred secondary group name. Click **Save**.

File Session Persistence

File persistence stores session information in files to a highly available file system.



Session state may also be stored in the file system.

For file-based persistence:

- You must create the directory in which to store the files.
- The servers must have the appropriate access privileges.

Any server can act as the secondary server to back up any primary server. Therefore, the session cookie does not keep track of a secondary server.

Configuring File Persistence

1. Create a folder shared by all servers on the cluster on a highly available file system.
2. Assign read/write privileges to the folder.
3. Configure file session persistence in the `weblogic.xml` deployment descriptor.

```
...
<session-descriptor>
   <persistent-store-type>file</persistent-store-type>
   <persistent-store-dir>/mnt/wls_share</persistent-store-dir>
</session-descriptor>
...
```

weblogic.xml

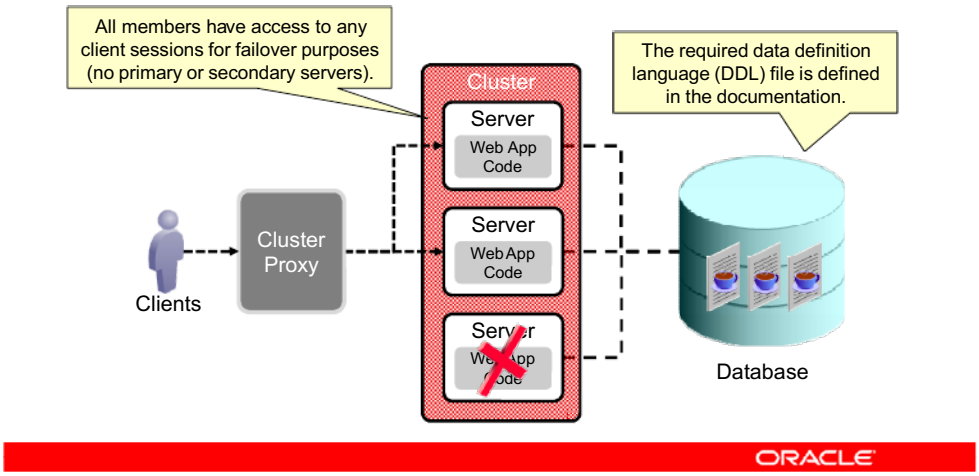
In the `weblogic.xml` deployment descriptor file, set the `persistent-store-type` parameter in the `session-descriptor` element to `file`.

Set the directory where WebLogic stores the sessions using the `persistent-store-dir` parameter. You must create this directory and make sure that appropriate access privileges are assigned to the directory.

Ensure that you have enough disk space to store the number of valid sessions multiplied by the size of each session. You can find the size of a session by looking at the files created in the location indicated by the `persistent-store-dir` parameter. Note that the size of each session can vary as the size of serialized session data changes.

JDBC Session Persistence

HTTP sessions are persisted to a database using a common JDBC data source.

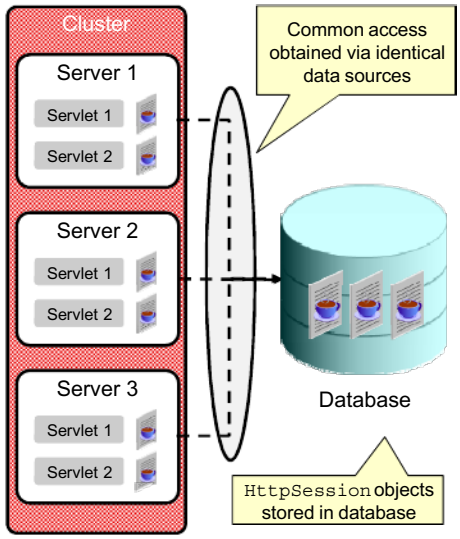


With Java Database Connectivity (JDBC) session persistence, a database is configured for storing `HttpSession` objects. After the database is configured, each server instance in a cluster uses an identical connection pool to share access to the database.

Whenever a web application creates or uses a session object, the WebLogic web container stores the session data persistently in the database. When a subsequent client request enters the cluster, any server in the cluster can handle the request. Each server in the cluster has identical access to the persistent store where it can look up the information needed to satisfy the client's request. This technique provides good failover capability because any server in the cluster can resolve a client's request, but there is a significant performance reduction due to the many database synchronizations required in a large web-based system. Because any server can respond to any request, the session cookie does not keep track of primary or secondary servers.

JDBC Session Persistence Architecture

- All server instances have access to all sessions.
- Subsequent requests from the same client can be handled by any server.
  - ✓ Great failover capability
  - ✗ Significant performance reduction
- Changing session objects causes (slow) database synchronization.



ORACLE

Whenever a servlet creates or uses a session object, the servlet stores the session data persistently in the database. When a subsequent client request enters the cluster, any server in the cluster can handle the request. Each server in the cluster has identical access to the persistent store where it can look up the information needed to satisfy the client's request. This technique provides for good failover capability because any server in the cluster can resolve a client's request, but there is a significant performance reduction due to the many database synchronizations required in a large web-based system.

Session persistence is not used for storing long-term data between sessions. That is, you should not rely on a session still being active when a client returns to a site at some later date. Instead, your application should record long-term or important information in a database.

You should not attempt to store long-term or limited-term client data in a session. Instead, your application should create and set its own cookies on the browser. Examples of this include an auto-login feature where the cookie lives for a long period or an auto-logout feature where the cookie expires after a short period of time. Here, you should not attempt to use HTTP sessions; instead you should write your own application-specific logic.

Note that even though it is legal (according to the HTTP Servlet specification) to place any Java object in a session, only those objects that are serializable are stored persistently by WebLogic.

Configuring JDBC Session Persistence

1. Create the required table in the database.
2. Create a JDBC data source that has read/write privileges for your database.
3. Configure JDBC session persistence in the `weblogic.xml` deployment descriptor.

```
...
<session-descriptor>
   <persistent-store-type>jdbc</persistent-store-type>
   <persistent-store-pool>mysessionds</persistent-store-pool>
</session-descriptor>
...
```

weblogic.xml

Set up a database table named `wl_servlet_sessions` for JDBC-based persistence. The connection pool that connects to the database needs to have read/write access for this table. Create indexes on `wl_id` and `wl_context_path` if the database does not create them automatically. Some databases create indexes automatically for primary keys.

Set the `persistent-store-type` parameter in the `session-descriptor` element in the `weblogic.xml` deployment descriptor file to `jdbc`.

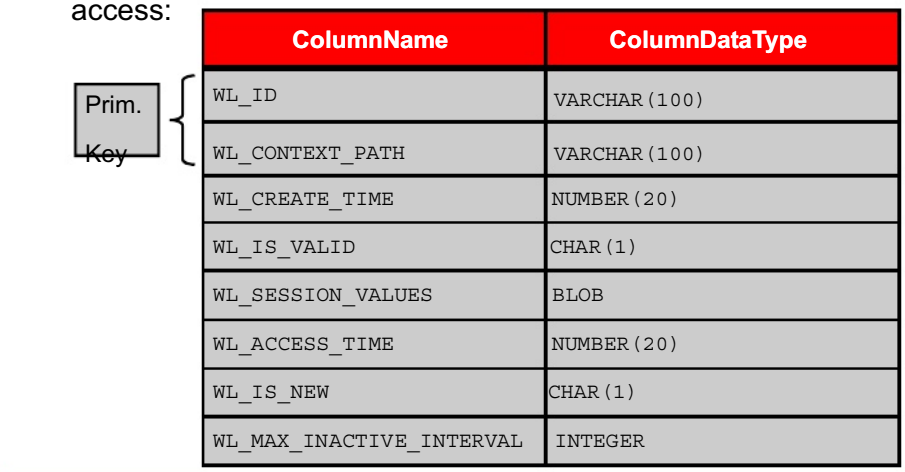
Set a JDBC connection pool to be used for persistence storage with the `persistent-store-pool` parameter in the `session-descriptor` element in the `weblogic.xml` deployment descriptor file. Use the name of a connection pool that is defined in the WebLogic administration console.

You can use the `jdbc-connection-timeout-secs` parameter to configure the maximum duration that the JDBC session persistence should wait for a JDBC connection from the connection pool, before failing to load the session data.

To prevent multiple database queries, WebLogic caches recently used sessions. Recently used sessions are not refreshed from the database for every request. The number of sessions in cache is governed by the `cache-size` parameter in the `session-descriptor` element of the WebLogic-specific deployment descriptor, `weblogic.xml`.

JDBC Persistent Table Configuration

The WL_SERVLET_SESSIONS table must exist with read/write access:



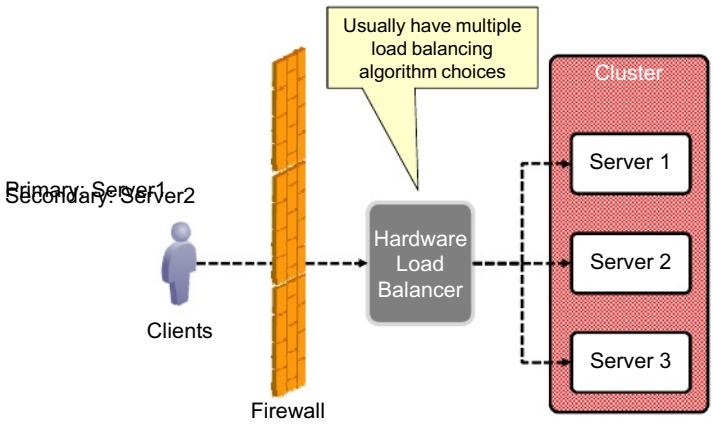
In the database that is referenced by the session persistence data source, you must configure a table, WL_SERVLET_SESSIONS, which will hold the session objects. The user specified with access to this table needs read/write/insert/delete access. The table columns are:

- WL_ID: The session ID
- WL_CONTEXT_PATH: This is the context. This column is used with WL_ID as the primary key. This is a variable-width alphanumeric data type of up to 100 characters.
- WL_IS_NEW: This value is true as long as the session is classified in the “new” state by the Servlet engine. This is a single char column.
- WL_CREATE_TIME: This is the time when the session was srcinally created. This is a numeric column, 20 digits.
- WL_IS_VALID: This parameter is true when the session is available to be accessed by a Servlet. It is used for concurrency purposes. This is a single char column.
- WL_SESSION_VALUES: This is the actual session data. It is a BLOB column.
- WL_ACCESS_TIME: This is the last time this session was accessed. This is a numeric column, 20 digits.
- WL_MAX_INACTIVE_INTERVAL: This is the number of seconds between client requests before the session is invalidated. It is an Integer. A negative value means the session should never time out.

The following is an example SQL statement to create this table, for Oracle Database:

```
CREATE TABLE "WL_SERVLET_SESSIONS"
(WL_ID VARCHAR (100)  NOT NULL,
 WL_CONTEXT_PATH VARCHAR (100) NOT NULL,
 WL_IS_NEW CHARACTER (1),
 WL_CREATE_TIME DECIMAL (20),
 WL_IS_VALID INTEGER,
 WL_SESSION_VALUES BLOB,
 WL_ACCESS_TIME DECIMAL (20) NOT NULL,
 WL_MAX_INACTIVE_INTERVAL INTEGER,
 PRIMARY KEY (WL_ID, WL_CONTEXT_PATH)
 );
```

Configuring a Hardware Load Balancer



Clusters that use a hardware load balancer can use any load balancing algorithm that is supported by the load balancer. If you choose to use load-balancing hardware instead of a proxy plug-in, you must use a hardware load balancer that supports secure sockets layer (SSL) persistence, passive cookie persistence, or active cookie persistence.

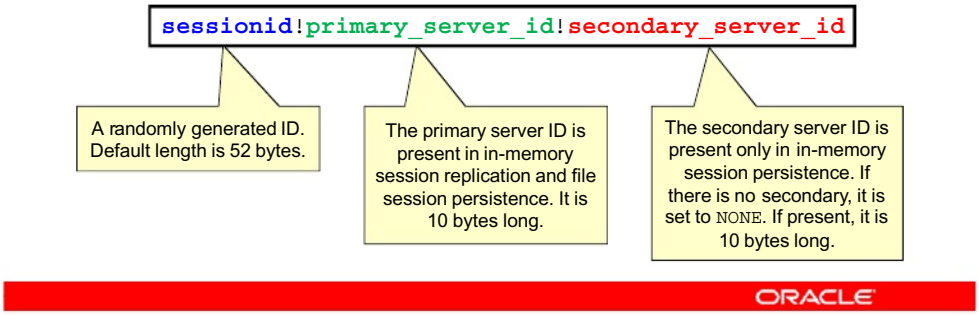
Hardware Load Balancer Session Persistence

- SSL Persistence
  - The load balancer performs all data encryption and decryption between clients and the WebLogic Server cluster.
  - The load balancer uses the plain text session cookie that WebLogic Server writes on the client to maintain an association between the client and the primary server
- Passive Cookie Persistence
  - The load balancer uses a string within the WebLogic Server session cookie to associate the client with the primary server. You must tell the load balancer where this string is.
- Active Cookie Persistence
  - If the load balancer creates it own cookie, and does not modify the WebLogic Server session cookie, this works without any additional configuration.

- If SSL persistence is used, then the load balancer performs all encryption and decryption of data between clients and the cluster. The load balancer then uses the plain-text cookie created by WebLogic Server to maintain the association between the client and the server in the cluster.
- Passive cookie persistence means the load balancer allows WebLogic Server to write its session cookie through the load balancer to the client. The load balancer, in turn, interprets an identifier in the client's cookie to maintain the relationship between the client and the primary WebLogic Server that hosts the HTTP session state.
- You can use certain active cookie persistence mechanisms with WebLogic Server clusters, provided the load balancer does not modify the WebLogic Server 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.

Passive Cookie Persistence and the WebLogic Server Session Cookie

- Configure a passive cookie load balancer:
  - Cookie name: `JSESSIONID`
  - Set the offset to `53` bytes (52 bytes for the session ID + 1 byte for the delimiter)
  - String length: `10` characters



The hardware load balancer works with all session persistence types because the behavior is the same in each case. When the primary is not available, the load balancer uses its configured algorithm to select another server in the cluster. The secondary ID is not really used by the load balancer, but when in-memory persistence is configured the server that receives the request uses the cookie to fetch the session from the secondary session. In the other session failover types, the secondary is not used at all.

To configure a load balancer to work with your cluster, configure the load balancer to define the offset and length of the string constant. The default length of the WebLogic Session ID portion of the session cookie is 52 bytes. Configure the load balancer to set the following:

- String offset to `53` bytes: This is the default random session ID length plus one byte for the delimiter character.
- String length to `10` bytes: This is the length of the identifier for the primary server.

Quiz

In-memory session replication copies session data from one clustered instance of WebLogic Server to:

a. All other instances of WebLogic Server in the cluster
b. All instances of WebLogic Server in the Preferred Secondary Group
c. All instances of WebLogic Server in the same Replication Group
d. Another instance of WebLogic Server in the cluster

Answer: d

Summary

In this lesson, you should have learned how to:

- Install Oracle HTTP Server
- Configure Oracle HTTP Server as a cluster proxy
- Configure session failover
- Configure replication groups

Practice 13-1 Overview: Installing OHS (Optional)

This practice covers the following topics:

- Installing OHS from the Web Tier installer
- Creating an OHS instance

Practice 13-2 Overview: Configuring a Cluster Proxy

This practice covers the following topics:

- Configuring Oracle HTTP Server to act as a proxy to a WebLogic cluster
- Starting Oracle HTTP Server
- Testing in-memory session replication

Practice 13-3 Overview: Configuring Replication Groups

This practice covers configuring replication groups in a cluster.

14 Clusters

Communication, Planning, and Troubleshooting

Objectives

After completing this lesson, you should be able to:

- Describe the differences between unicast and multicast cluster communication
- Configure a replication channel for a cluster
- Describe planning for a cluster
- Monitor and troubleshoot a cluster

Review: Cluster Communication

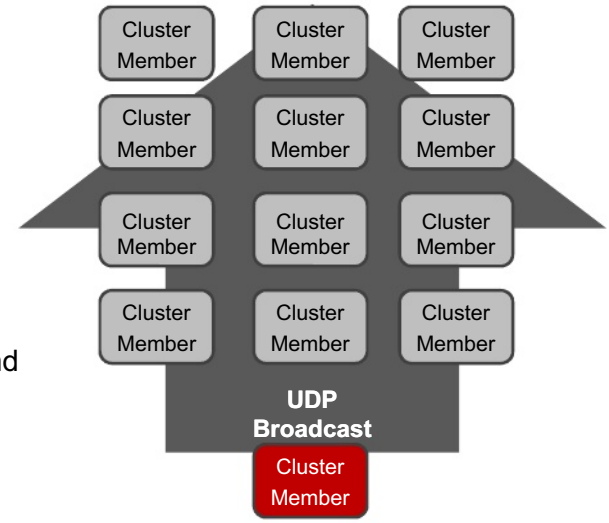
- Cluster members communicate with each other in two ways:
  - One-to-many messages
    - For periodic “heartbeats” to indicate continued availability
    - To announce the availability of clustered services
    - **Note:** This communication can use either:
      - IP unicast (recommended): No additional configuration is required.
      - IP multicast: A multicast host and port must be configured.
  - Peer-to-peer messages
    - For replicating HTTP session and stateful session EJB state
    - To access clustered objects that reside on a remote server (multi-tier architecture)
    - **Note:** This communication uses sockets.

An instance of WebLogic Server uses one-to-many communication to send regular “heartbeat” messages that advertise its continued availability to other server instances in the cluster. The servers in a cluster listen for heartbeat messages to determine when a server has failed.

All servers use one-to-many messages to announce the availability of clustered objects that are deployed or removed locally. Servers monitor these announcements so that they can update their local JNDI tree to indicate the current deployment of clustered objects. This is the maintenance of the so-called “cluster-wide” JNDI tree.

How Multicast Works

Oracle does not recommend using multicast communication and supports it only for backward compatibility.



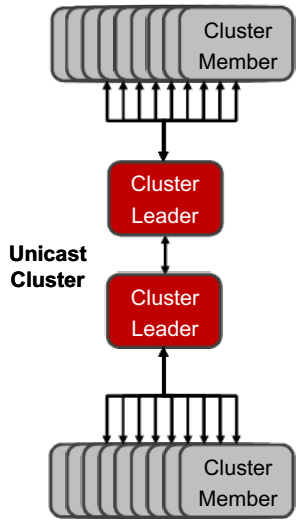
IP multicast enables multiple applications to subscribe to an IP address and port number, and listen for messages. A multicast address is an IP address in the range `224.0.0.0-239.255.255.255`. IP multicast does not guarantee that messages are received, so WebLogic Server allows for the possibility that some messages may be missed. If you use multicast, you must ensure your network propagates multicast messages to all clustered servers. The multicast time-to-live value can be increased if you find that messages are being missed. With multicast, you must ensure that no other applications share the multicast address and port, or servers will have to process extra messages, which introduces extra overhead.

Firewalls can break multicast transmissions. Although it might be possible to tunnel multicast transmissions through a firewall, this practice is not recommended. A final worry with multicast messaging is the possibility of a multicast "storm," in which server instances do not process incoming messages in a timely fashion, which leads to retransmissions and increased network traffic.

How Unicast Works

Unicast messaging:

- Uses TCP-IP networking
- Creates a connection for each server
- Uses a hub-and-spoke design so that it scales
- Divides a cluster into groups and assigns a group leader to each
- Enables group leaders to manage communication between groups



ORACLE

The TCP protocol requires a point-to-point connection. Unlike multicast, which broadcasts to many servers simultaneously, unicast needs to create a connection for each server in the cluster. WLS would not scale well if every server in a cluster had to connect to every other server in the cluster. WebLogic implements unicast to scale well. When a cluster starts, the servers divide the cluster into groups of ten cluster members. One member in each group becomes the group leader. This creates a network topology that reduces the number of connections an individual member makes with other members in the cluster.

The picture in this slide shows a twenty-member unicast cluster. WebLogic Server divides the cluster into two groups of ten servers, and each group elects a leader. Group members send and receive cluster messages through their group leader, and group leaders communicate with each other to make cluster traffic scalable. Group leaders act as simple network relays to their group members, and to other group leaders. Group leaders and members can receive multiple messages because they do not store any state data, so there is no risk of data corruption.

If a group leader is not available, another group member becomes the new group leader. If the srcinal group leader becomes available again, the old group leader becomes the group leader again, and the cluster demotes the acting group leader back to a regular group member.

Unicast Versus Multicast

Unicast communication is preferred for the following reasons:

- TCP-IP is the defacto standard protocol of the Internet.
- Many companies do not support multicast in production.
- Several networking devices do not support multicast.
- Unicast is easier to configure and reduces traffic.

IP unicast is recommended because it does not have the network issues of multicast. You can set up a separate network channel for unicast communication, but it is not required. If no separate channel is defined, each server's default channel is used (the default channel is the server's configured host and port).

Configure Multicast

First, you should test if the multicast address you want to use is working using the `MulticastTest` tool.

You can verify that multicast is working by running `utils.MulticastTest` from one of the Managed Servers.

The `MulticastTest` utility helps you to debug multicast problems when configuring an Oracle WebLogic Server cluster. The utility sends out multicast packets and returns information about how effectively the multicast is working on your network. Specifically, `MulticastTest` displays the following types of information via standard output:

1. A confirmation and sequence ID for each message sent out by the current server
2. The sequence and sender ID of each message received from any clustered server, including the current server
3. A missed-sequenced warning when a message is received out of sequence
4. A missed-message warning when an expected message is not received

To use `MulticastTest`, start one copy of the utility on each node on which you want to test the multicast traffic.

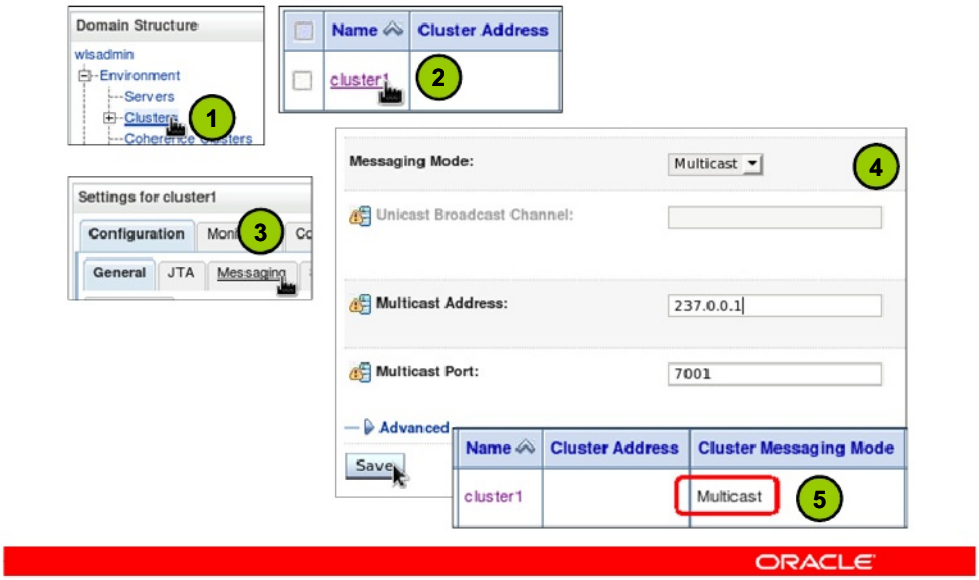
Warning: Do not run the `MulticastTest` utility by specifying the same multicast address (the `-a` parameter) as that of a currently running Oracle WebLogic Server cluster. The utility is intended to verify that the multicast is functioning properly before your clustered Oracle WebLogic Servers are started.

Syntax

```
$ java utils.MulticastTest -n name -a address [-p portnumber]
[-t timeout] [-s send]
```

- `-n name` (required): A name that identifies the sender of the sequenced messages. Use a different name for each test process that you start.
- `-a address`: The multicast address on which: (a) the sequenced messages should be broadcast; and (b) the servers in the clusters are communicating with each other. (The default is 237.0.0.1.)
- `-p portnumber` (optional): The multicast port on which all the servers in the cluster are communicating. (The multicast port is the same as the listen port that is set for Oracle WebLogic Server, which defaults to `7001` if unset.)
- `-t timeout` (optional): Idle timeout, in seconds, if no multicast messages are received. If unset, the default is `600` seconds (10 minutes). If a timeout is exceeded, a positive confirmation of the timeout is sent to `stdout`.
- `-s send` (optional): Interval, in seconds, between sends. If unset, the default is `2` seconds. A positive confirmation of each message that is sent out is sent to `stdout`.

Configure Multicast

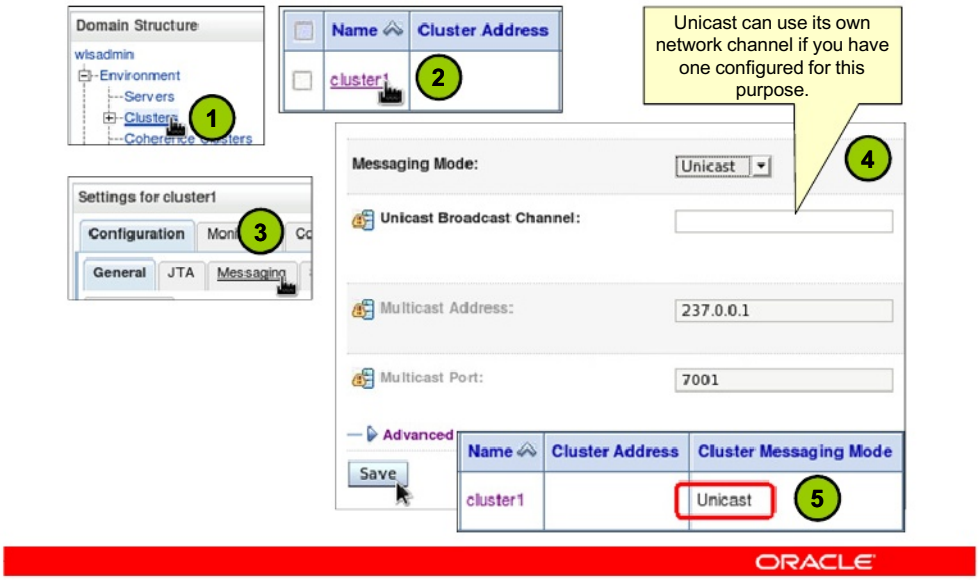


You configure clustering using the following steps:

1. In the administration console, expand **Environment** and select **Clusters**.
2. Select the cluster you want to configure to use multicast communication.
3. Select the **Configuration** > **Messaging** tabs to display the cluster's broadcast communication settings page.
4. Set the Messaging Mode to `Multicast`, set the Multicast IP address, and Multicast Port. In this case, `237.0.0.1` is configured as the IP address and `7001` as the port. Save your changes.
5. Review your cluster and see that its Cluster Messaging Mode is now Multicast.

Note: This change requires restarting the affected servers of the cluster. Note that when you select the multicast messaging mode, the unicast settings are unavailable.

Configure Unicast



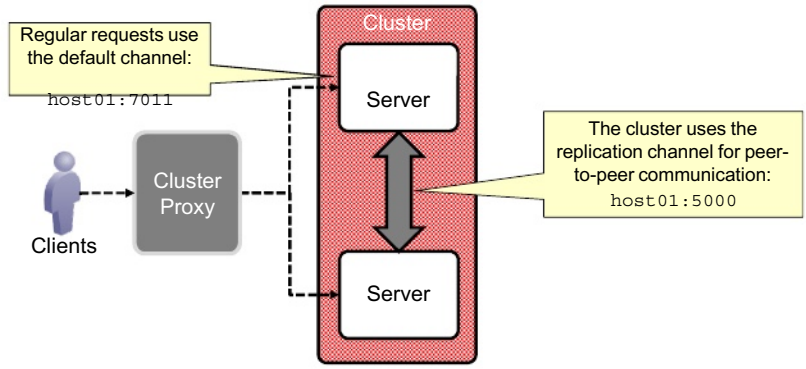
You configure clustering using the following steps:

1. In the administration console, expand **Environment** and select **Clusters**.
2. Select the cluster you want to configure to use unicast communication.
3. Select the **Configuration** > **Messaging** tabs to display the cluster's broadcast communication settings page.
4. Set the Messaging Mode to `Unicast`. There is also a field for entering the name of a network channel if you want to have unicast traffic on its own channel. In this case, we are just allowing traffic to use the default channel. Save your changes.
5. Review your cluster and see that its Cluster Messaging Mode is now `Unicast`.

Note: This change requires restarting the affected servers of the cluster. Note that when you select the unicast messaging mode, the multicast settings are unavailable.

Replication Channel

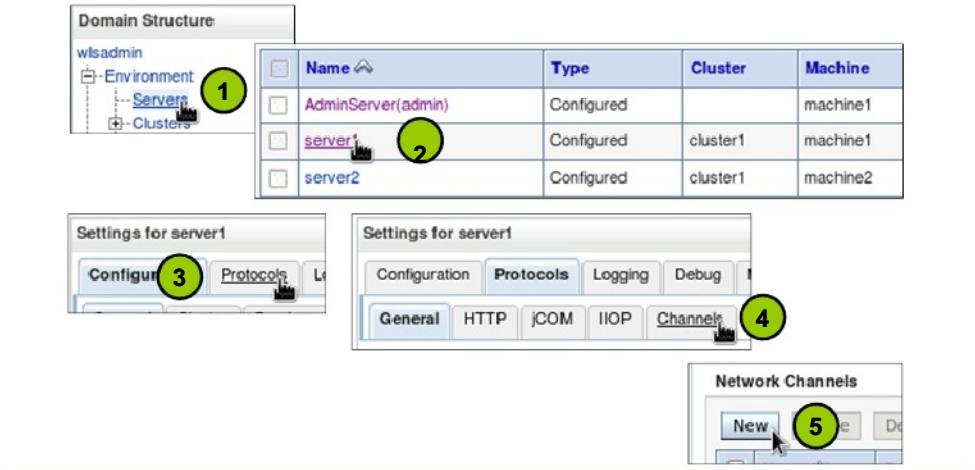
WebLogic Server allows you to configure a separate network channel for peer-to-peer cluster communication (replication).



In a corporate enterprise environment, network volumes can reach tremendous levels that may saturate a network. If this network is the same one that the WebLogic default channel uses for its servers, this can hinder high-priority or internal traffic. One example of this is session replication. Some applications may be more replication-intensive than others and can benefit from separating replication traffic from other traffic in the server. In other scenarios, such as when using Exalogic InfiniBand, some network stacks offer much higher performance than standard TCP-IP networks. WebLogic applications can benefit from using a faster network for replication if there is a lot of replication traffic. This allows client traffic and replication traffic to operate on different networks to avoid saturating the network.

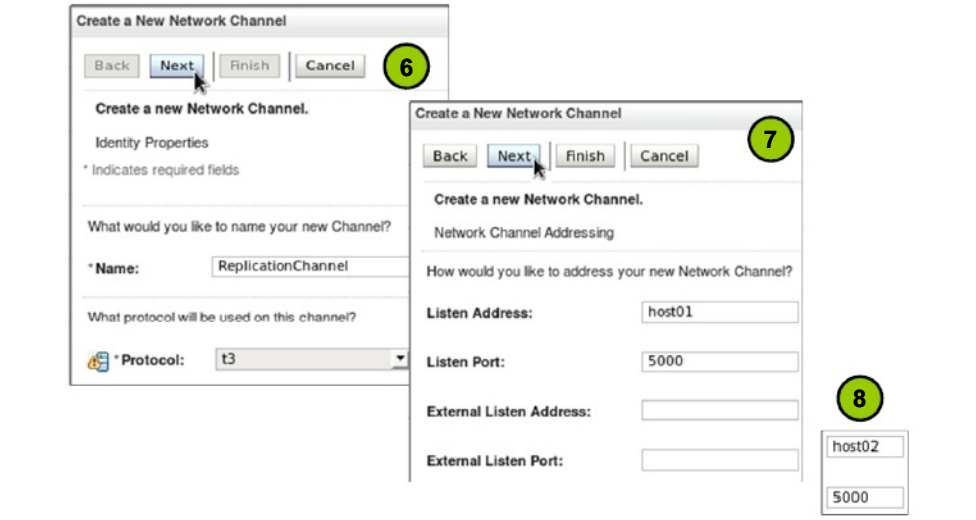
Configure Replication Channels: Servers

First, configure each server with a network channel:



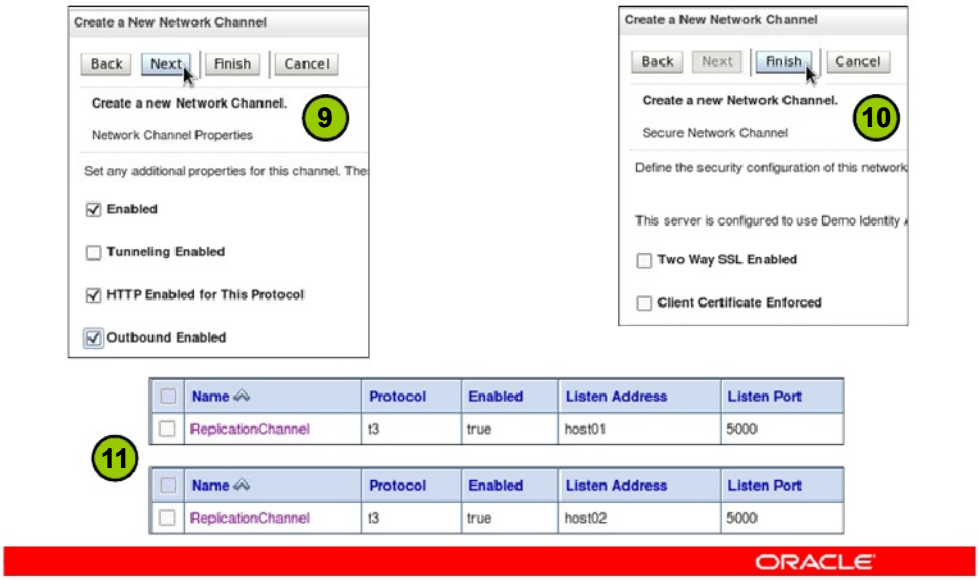
1. Within the administration console, expand **Environment** and select **Servers**.
2. Select the server for which you want to create a channel.
3. Select the **Protocols** tab.
4. Select the **Channels** subtab to display the list of configured channels for this server.
5. Click **New** to create a new channel.

Configure Replication Channels: Servers



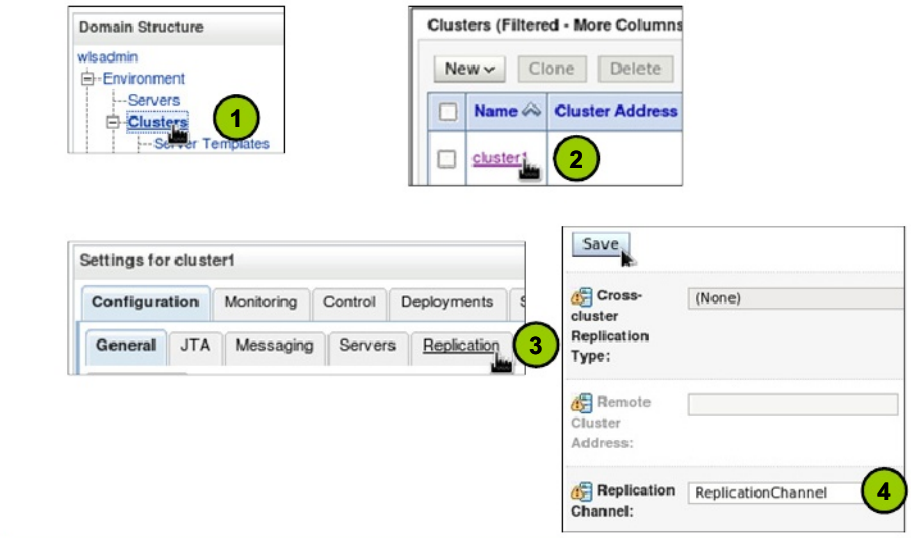
6. Enter a name and select a protocol for your channel. In this case, the name is `ReplicationChannel` and the protocol is `t3`. Click **Next**.
7. Configure the network addressing for your channel. In this case, the liste n address is `host01` and the listen port is `5000`. Click **Next**.
8. This procedure is repeated for each server in the cluster. Ensure that the network channel has the same name for each server. Assuming a two-node cluster for this example, this procedure is repeated for the second server with a listen address and port of `host02` and `5000`, respectively.

Configure Replication Channels: Servers



9. Next, you configure the properties for your channels. In this example, the channel is enabled, HTTP is enabled, and it allows for outbound communication. A replication channel must allow for outbound communication so it can both send and receive replication messages. Click **Next**. Do the same thing for each of the cluster server channels.
10. If there are any SSL requirements, you configure them on this page. In this example, you are not using SSL so you just click **Finish**.
11. After your network channels are created for all the servers in your cluster, you can view them in the console. Now you have to configure your cluster to use this channel for replication.

Configure Replication Channels: Cluster



1. Within the administration console, expand **Environment** and select **Clusters**.
2. Select the cluster you want to configure to use the new replication channel.
3. Select the **Replication** subtab under the **Configuration** tab.
4. Enter the name of the replication channel for this cluster to use. In this example, the same name that was used when creating the channels on each server in the cluster, **`ReplicationChannel`** is used. This tells the cluster to use the channel named `ReplicationChannel` for all replication traffic.

Note: You can optionally use SSL to secure your replication channel; however, doing so can potentially cause a slowdown in performance because most replication is done synchronously. This means that when a client updates its session state, that client waits for WebLogic to finish updating the state of the secondary session before getting control back.

Configure Replication Channels

You can verify that your replication channel is enabled by checking the system out of each server in the cluster.

```
<Notice> <Server> <BEA-002613> <Channel " ReplicationChannel"
is now listening on 192.0.2.11:5000 for protocols t3, CLUSTER-
BROADCAST, http.>
```

server1 output

```
<Notice> <Server> <BEA-002613> <Channel " ReplicationChannel"
is now listening on 192.0.2.12:5000 for protocols t3, CLUSTER-
BROADCAST, http.>
```

server2 output

Planning for a Cluster

1. Determine your cluster architecture.
   - Basic
   - Multi-tier
2. Consider your network and security topologies.
   A. Where to place firewalls
      - Do not place firewalls in between cluster members.
   B. Decide on one-to-many cluster communication
      - Multicast
      - Unicast
3. Determine the type of cluster you will define.
   - Regular cluster
   - Dynamic cluster

Planning for a Cluster

4. Choose hosts for the cluster.
   A. Note each host's DNS name or IP address. DNS or virtual host names are recommended.
   B. Choose the port number for each managed server*. Note the admin server host and port.
   C. Decide on the names of servers*, machines, clusters, and so on (each WebLogic resource must have a unique name).
   D. Start with one managed server per CPU core.
      - You can scale up later based on performance testing.
5. Choose your cluster proxy
   - Web server with a proxy plug-in
   - Hardware load balancer

* With Dynamic Clusters, some of these values are generated. For example, the server name prefix or starting port number is defined.

For production environments, the use of DNS names is generally recommended. Virtual host names could also be used. The use of IP addresses can result in translation errors if:

- Clients connect to the cluster through a firewall, or
- You have a firewall between the web application and EJB tiers

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.

Oracle recommends that you start with one server per CPU core and then scale up based on the expected behavior. You should test the actual deployment with your system to determine the optimal number and distribution of server instances.

Planning for a Cluster

6. Decide how to handle HTTP session failover.
   - In-memory replication
   - File storage
   - JDBC storage
   - Coherence*Web
7. If using the multi-tier architecture with EJBs, decide on the EJB load balancing algorithm.
   - Round-robin
   - Random
   - Weight-based
8. Decide how pinned services will be handled.
   - Service-level migration
   - Whole server migration

EJB load balancing choices are:

- Round-robin (the default): The algorithm cycles through a list of WebLogic Server instances in order. The advantages of the round-robin algorithm are that it is simple, quick, and very predictable. The disadvantage is it treats all servers the same (even though you may have some that are faster than others).
- Random: The algorithm routs requests to servers at random. The 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.
- Weight-based: Each server hosting EJBs can be given a weight. Select the server in the Servers table. Select **Configuration** > **Cluster**. Enter a number between `1` and `100` in the **Cluster Weight** field. This value determines what proportion of the load the server will bear relative to other servers in the EJB cluster.

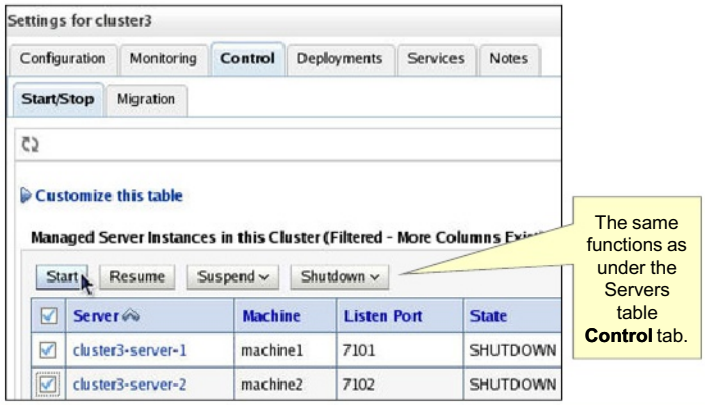
A pinned service is one that is active on only one cluster host. JTA (transaction) recovery and JMS servers are pinned services.

Service-level migration is migrating a pinned service from a failed cluster member to one that is active. It can be done manually or occur automatically.

Whole-server migration is an entire instance of WebLogic Server migrated to a different physical machine upon failure. It, too, can be done manually or happen automatically.

Managing a Cluster

Select the cluster in the Clusters table and click the **Control** tab. The **Start/Stop** subtab shows the servers in the cluster and allows you to start, stop, suspend, and resume them.



The Migration subtab allows you to manually migrate “singleton” services from one server in the cluster to another. Service-level and whole server migration are covered in the *Oracle WebLogic Server 12c: Administration II* course.

Troubleshooting a Cluster

When there are issues with a cluster, you have tools to help:

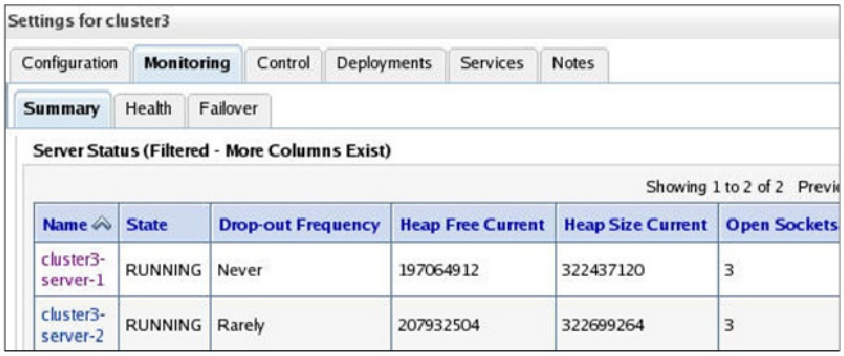
- WebLogic Server logs
- OHS logs
- Monitoring by using the administration console or the Monitoring Dashboard
- WLST

Common problems include:

- OHS to WebLogic Server connectivity issues
- Multicast communication issues (if using multicast)
- Cluster member uniformity problems
- Session failover issues

Monitoring a Cluster: Admin Console

- In the administration console: Select the cluster from the Clusters table, use its Monitoring tab and its subtabs.



The table was customized to show each cluster server: name, state, how often it left the cluster (went down), free heap memory, total heap size, and number of open sockets.

WebLogic Server and OHS Logs

- The WebLogic Server logs contain cluster subsystem messages.
  - Set debug flags to generate more detailed log messages:
    - In the administration console, select the server from Servers table, click the **Debug** tab, expand scopes, and select flags.
- The OHS logs are found here: `<ORACLE_INSTANCE>/diagnostics/logs/OHS/`*`OHS_instance_name`*
  - The OHS error log: *`<OHS_INSTANCE_NAME>`*`.log`
    - Records OHS errors, but can be configured to record events.
  - The OHS access log: `access_log`
    - Records which components and applications are accessed and by whom

There are cluster debug flags in the `weblogic.core.cluster`, `weblogic.cluster`, and `weblogic.servlet.internal.session` scopes.

There are two types of logs for Oracle HTTP Server. Error logs record server problems, but can also be configured to record other server events. Access logs record which components and applications are being accessed and by whom. The location of the OHS logs is configurable, as are the names of the log files. The location and names given in the slide are their defaults.

Common OHS to WLS Connectivity Issues

- Connectivity problems can cause unnecessary failovers or HTTP errors to be sent to the client.
- Causes of unexpected connection failures include problems with these OHS parameters:
  - `WebLogicCluster` (the initial list of cluster members)
    - If this list is incorrect, the plug-in may not be able to proxy.
  - `ConnectTimeoutSecs` (how long the plug-in waits to establish a connection)
    - If this is set too low, the plug-in can give up on a server and not connect to it.
  - `ConnectRetrySecs` (pause time before retrying a connection)
    - If this is accidentally set higher than `ConnectTimeoutSecs`, the plug-in will always time out during a retry.

Common OHS to WLS Connectivity Issues

- Causes of unexpected request failures include problems with these OHS parameters:
  - `WLIOTimeoutSecs` (the amount of time the plug-in waits for a response from WebLogic Server)
    - If this is set too low, and WebLogic Server sometimes takes a long time to process a request, that server will be considered dead by OHS, even though it is not.
  - `MaxPostSize` (the size of a post)
    - If this is set too low on either the proxy or on the WebLogic Server instance, a request can fail because the request is too large.
  - `MaxSkipTime` (the wait time before the plug-in retries a server marked as "bad")
    - If this is set too high, the proxy will be slow to use a restarted cluster member, affecting overall performance.

The `WLIOTimeoutSecs` parameter should typically be set to a large value (the default is five minutes). If the value is less than the time your application takes to process a request, then you may see unexpected results.

If the `MaxPostSize` parameter is greater than or equal to the same WLS setting, it will have no effect. The Max Post Size setting for an instance of WebLogic Server can be found in the admin console under a server's **Protocols** > **HTTP** tabs.

Multicast Communication Issues

- Problem with the multicast address
  - For each cluster on the network, the combination of the multicast address and port must be unique.
  - Ensure no other applications use that address and port.
- Missed multicast messages (heartbeats) can cause cluster members to be marked as “failed.”
  - Ensure the multicast time to live (TTL) value is large enough for the messages to get to all cluster members.
  - If multicast buffers fill up, messages are missed.
    - Increase the size of the multicast buffer.
    - Increase the multicast send delay, which helps avoid buffer overflow.

One reason unicast is recommended is that there are generally less issues with it.

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.
- 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.
- The value of the multicast time-to-live (TTL) parameter for the cluster must be high enough to ensure that routers do not discard multicast packets before they reach their final destination.
- Increasing the size of the multicast buffers can improve the rate at which announcements are transmitted and received, and prevent multicast storms. (A multicast storm is the repeated transmission of multicast packets on a network. Multicast storms can stress the network, potentially causing end-stations to hang or fail.)
- 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.

Cluster Member Uniformity

- Every instance of WebLogic Server in a cluster should be like every other. All servers should:
  - Be the same version of WebLogic Server
  - Have the same `CLASSPATH`
  - Have the same deployments
  - Have the same services (like data sources)
- When cluster members are not the same, you have intermittent problems, which are very hard to debug.

As an example, let us say five of the six cluster servers have the data source targeted to them. When clients are routed to five of the servers in the cluster, there are no problems when the application tries to use the database. When a client is routed to the sixth server, however, the application gives them errors, because the data source cannot be found.

Session Failover Issues

- Session replication or persistence problems often result in the loss of session data. This affects your clients:
  - A client must log in again.
  - The client's shopping cart items disappear.
- Typical culprits include:
  - Invalid session persistence settings
  - Session or cookie timeout settings are too low.
  - The developers of the web application did not use the `HttpSession` API appropriately.
  - The developers of the web application are storing non-serializable objects in the session.
    - Objects must be serializable so they can be streamed from the primary server to the secondary server (in-memory replication) or to files or the database (file or database persistence).

To serialize an object means to convert its state to a byte stream so that the byte stream can be reverted into a copy of the object. A Java object is serializable if its class or any of its superclasses implements either the `java.io.Serializable` interface or its subinterface, `java.io.Externalizable`

Quiz

A replication channel is:

a. Another name for replication group
b. Another name for the preferred secondary group
c. A network channel used by cluster members for peer-to-peer communication
d. The title of the tab in the Monitoring Dashboard that shows cluster charts

Answer: c

Summary

In this lesson, you should have learned how to:

- Describe the differences between unicast and multicast cluster communication
- Configure a replication channel for a cluster
- Describe planning for a cluster
- Monitor and troubleshoot a cluster

Practice 14-1 Overview: Configuring a Replication Channel

This practice covers configuring a replication channel for a cluster.

15

Transactions

Objectives

After completing this lesson, you should be able to:

- Describe WebLogic Server's role in managing transactions
- Configure a database persistent store for transactions

Transactions and ACID

- A transaction is a mechanism to handle a group of operations as if they were one. It is a unit of work.
- Transactions have four key properties (ACID).
  - **A**tomic: The entire sequence of operations must either be completed successfully or be as if none of them occurred at all. The transaction cannot be partially successful.
  - **C**onsistent: A transaction transforms a system from one valid state to another valid state.
  - **I**solated: Each transaction occurs independently. Its effect is not visible until it has completed.
  - **D**urable: Completed transactions remain permanent, even during system failure.

Why are transactions needed? Suppose a client application needs to make a service request that might involve write operations to multiple databases. If any one invocation is unsuccessful, any state that is written (either in memory or, more typically, to a database) must be rolled back. For example, consider an interbank fund transfer application in which money is transferred from one bank to another. The transfer operation requires the server to perform the following tasks:

1. Call the withdraw method on an account at the first bank.
2. Call the deposit method on an account at the second bank.

If the deposit method at the second bank fails, the banking application must roll back the previous withdrawal at the first bank.

Transactions should have the following ACID properties:

- **Atomic:** All or nothing. All operations involved in the transaction are completed successfully or none are completed at all.
- **Consistent:** The database or other resource must be modified from one valid state to another. In the event the system or database fails during the transaction, the srcinal state is restored (rolled back).
- **Isolated:** An executing transaction is isolated from other executing transactions. A transaction's effects cannot be seen until it has completed.
- **Durable:** After a transaction is committed, it can be restored to this state in the event of a system or database failure.

Global Transactions, 2PC, and XA

- A global (distributed) transaction involves more than one transactional resource.
  - A transaction manager (TM) deals with each resource manager (RM).
- The Two-Phase Commit (2PC) protocol uses two steps to commit changes within a global transaction:
  - Phase 1: TM asks RMs to prepare to make the changes.
  - Phase 2: If all RMs report that they are ready to commit, TM tells the RMs to commit, which makes the changes permanent. If any RM is not ready to commit, TM tells all RMs to roll back (undo any changes).
- The Extended Architecture (XA) specification implements the 2PC protocol.

WebLogic Server can act as a TM.

Some terms associated with transactions:

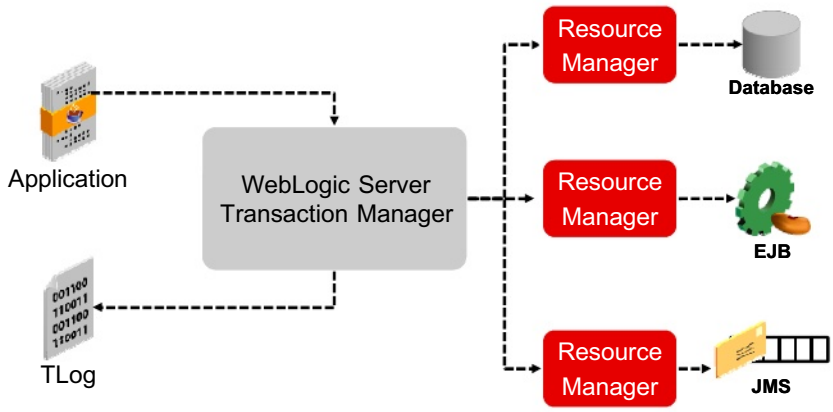
- A resource, like a database, is controlled through software called a *resource manager* (RM).
- A *transaction manager* (TM) coordinates multiple resource managers.
- A transaction manager manages transactions on behalf of application programs. A transaction manager coordinates commands from the application programs to start and complete transactions by communicating with all the resource managers that are participating in those transactions. When resource managers fail during transactions, transaction managers help resource managers decide whether to commit or roll back pending transactions.
- A recoverable resource provides persistent storage for data. The resource is typically a database.
- A resource manager provides access to a collection of information and processes.

  Transaction-aware JDBC drivers are common resource managers.

The Extended Architecture (XA) specification comes from the Open Group (`http://www3.opengroup.org`), a global consortium that works on IT standards.

WebLogic Server as a Transaction Manager

WebLogic Server coordinates a global transaction with the various transactional resource managers involved.

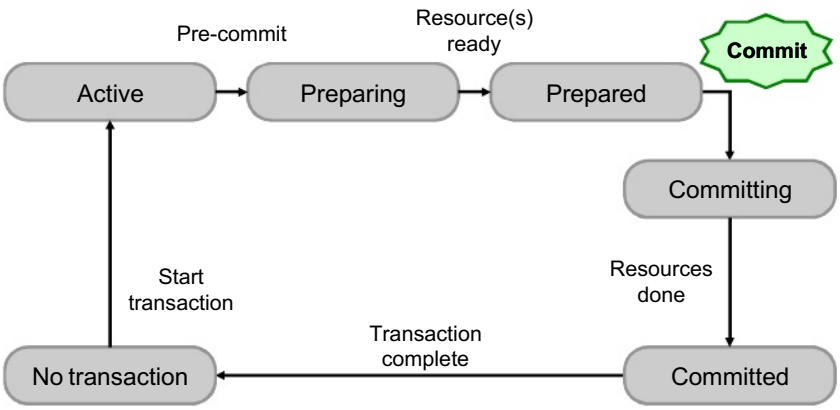


WebLogic Server implements the Java Transaction API (JTA) to manage transactions.

It can act as the transaction manager to the various transactional resource managers in a global, or distributed, transaction.

As it coordinates a global transaction, it tracks the transaction in a binary transaction log (also called a TLog).

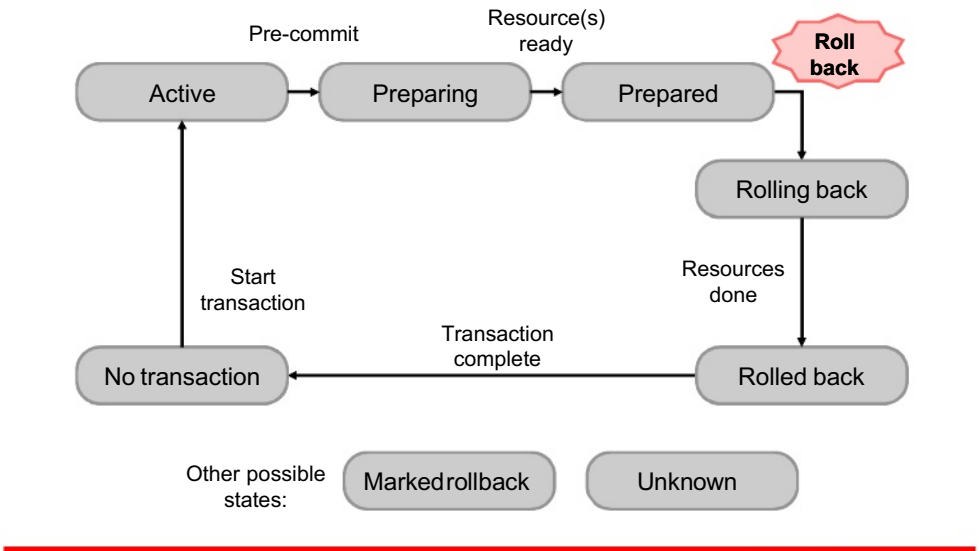
Transaction States when Committing



States:

- **No transaction:** No transaction in progress
- **Active:** The application is processing the transaction. The transaction has not yet reached the two-phase commit processing.
- **Preparing:** In the first phase of 2PC before all participants have responded: "ready to commit."
- **Prepared:** In between when all participants have responded to "prepare" but before the commit point or the initiation of rollback processing
- **Committing:** The time from when the commit decision is made up to the point when all participants have been informed of the outcome and the commit is complete
- **Committed:** The transaction has been committed

Transaction States when Rolling Back



States:

- **No transaction:** No transaction in progress
- **Active:** The application is processing the transaction. The transaction has not yet reached the two-phase commit processing.
- **Preparing:** In the first phase of 2PC before all participants have responded: "ready to commit"
- **Prepared:** In between when all participants have responded to "prepare" but before the commit point or the initiation of rollback processing
- **Rolling back:** This state occurs from the point when rollback processing is initiated up to the point when all participants have been instructed to roll back and the rollback is complete.
- **Rolled back:** The transaction has been rolled back.
- **Marked rollback :** The transaction has been marked for rollback by application code, as with the `setRollbackOnly()` method. The transaction has not started to roll back, but it will be rolled back, eventually.
- **Unknown:** The transaction is in a transient condition. Currently, WebLogic Server does not know the state of the transaction. The state will change soon.

Java Transaction API (JTA)

- WebLogic Server uses JTA to implement and manage global transactions.
- WebLogic Server's JTA implementation:
  - Creates a unique transaction identifier (`XID`)
  - Supports an optional transaction name
  - Tracks objects involved in transactions
  - Notifies databases of transactions
  - Orchestrates 2PC using XA
  - Executes rollbacks
  - Executes automatic recovery procedures in the event of failure
  - Manages timeouts

WebLogic Server's implementation of JTA provides the following support for transactions. It:

- Creates a unique transaction identifier when a client application initiates a transaction.
- Supports an optional transaction name describing the business process that the transaction represents. The transaction name makes statistics and error messages more meaningful.
- Works with the WebLogic Server infrastructure to track objects that are involved in a transaction and, therefore, must be coordinated when the transaction is ready to commit.
- Notifies the resource managers (typically, databases) when they are accessed on behalf of a transaction. Resource managers lock the accessed records until the end of the transaction.
- Orchestrates the two-phase commit when the transaction completes, which ensures that all the participants in the transaction commit their updates simultaneously. WebLogic Server coordinates the commit with any databases that are being updated using the Open Group's XA protocol. Most relational databases support this standard.

Configuring Transactions

To configure transactions for the domain, perform the following:

1. In the Change Center, click **Lock & Edit**. In the Domain Structure, select the domain name.
2. Select the **Configuration** tab and the **JTA** subtab.
3. Set the JTA attributes.
4. Click the **Save** button. In the Change Center, click **Activate Changes**.

Note that the monitoring of JTA and the JTA logging attributes are not set here. These are found at the server level.

JTA Configuration Options

Field	Description
Timeout Seconds	The time the TM waits for a new transaction to go into the prepared state
Abandon Timeout Seconds	The time the TM waits for a transaction to go from prepared to committed
Before Completion Iteration Limit	The maximum number of times the TM will call the `beforeCompletion()` method. This allows interested objects to take part in notifications as the transaction progresses.
Max Transactions	The maximum number of simultaneous in-progress transactions allowed on a server
Max Unique Name Statistics	The maximum number of unique transaction names for which statistics will be maintained
Checkpoint Interval Seconds	How often the TM creates a transaction log and checks to see if old logs can be deleted

- **Timeout Seconds:** How long the transaction manager will wait for the prepared state. If the timeout occurs and the transaction is still not prepared, it is rolled back.
- **Abandon Timeout Seconds:** How long the transaction manager will wait for the second phase of the transaction (to go from prepared to committed). If the timeout occurs, and the transaction is still in the prepared state, it is rolled back.
- **Before Completion Iteration Limit:** "Before completion" is the time for all resources to get to the prepared state. This attribute determines how many times the transaction manager will call the `beforeCompletion()` method. This is part of "synchronization," which allows resources to be notified before and after the transaction completes. A transactional resource can call another transactional resource, so it is sometimes necessary to call this method multiple times, as new objects are "registered" with the transaction manager. This attribute lets you limit the number of times this can occur.
- **Max Transactions:** The maximum number of simultaneous, in-progress transactions allowed on a server in this domain
- **Max Unique Name Statistics:** The maximum number of named transactions for which statistics will be maintained
- **Checkpoint Interval Seconds:** The interval at which the transaction manager creates a new transaction log and checks old logs to see whether they are ready to be deleted

JTA Configuration Options

Field	Description
Forget Heuristics	Whether or not the TM will automatically perform a "forget" operation for resources reporting a heuristic decision. The default is true. Disable this only if you know what to do with resources reporting heuristic decisions.
Unregister Resource Grace Period	The seconds the TM waits for transactions to complete before unregistering a resource (for example, when a data source is undeployed. it is unregistered). If at that time transactions are still outstanding, a log message is written.
Execute XA Calls in Parallel	XA calls are executed in parallel if threads are available.
Enable Two Phase Commit	Use 2PC for global transactions.

- **Forget Heuristics:** When enabled, the transaction manager automatically performs an XA Resource `forget()` operation for all resources as soon as there is a heuristic decision. A heuristic decision occurs when a resource unilaterally decides, during the completion stage of a transaction, to commit or rollback updates, no matter what it was instructed to do by the transaction manager. This can leave data in an inconsistent state. Network failures or resource timeouts are possible causes for heuristic decisions. Disable this feature only if you know what to do with the resource when it reports a heuristic decision.
- **Unregister Resource Grace Period:** The amount of time, in seconds, a transaction manager waits for the transactions involving the resource to complete before unregistering a resource. An example of a resource being unregistered is when you undeploy a data source. This grace period helps minimize the risk of abandoned transactions because of an unregistered resource. At the end of the grace period, if outstanding transactions are associated with the resource, a log message is written to the server on which the resource was previously registered.
- **Execute XA Calls in Parallel:** If threads are available, execute XA calls in parallel. This is enabled by default.

- **Enable Two Phase Commit:** Indicates that the two-phase commit protocol is used to coordinate transactions across two or more resource managers. If not selected, the two-phase commit protocol is disabled and any attempt to use two-phase commit results in a `RollbackException`being thrown. Also, if not selected, all transaction logging is disabled. This attribute is enabled by default.

Note: There are more JTA options that are not shown.

WebLogic Extension of JTA

Developers writing transactional code to run on WebLogic Server have available WebLogic-specific extensions to standard JTA.

- The WebLogic JTA transaction object supports the `weblogic.transaction.Transaction` interface (which extends `javax.transaction.Transaction`).
  - This adds various capabilities, the most important of which, to an administrator, is the ability to name transactions.
    - If developers write their code well, transactions can have business names (not just transaction IDs), which makes statistics and error messages more meaningful.

For more information about the WebLogic extensions to JTA, see the *Oracle WebLogic Server API Reference*. Look for the `weblogic.transaction` package.

JDBC Reminder

- For your database to participate in global transactions, choose an XA driver when creating the data source.
- If you must choose a non-XA driver, select **Supports Global Transactions**, and then select how the driver will support them. The recommendation is **Logging Last Resource**.
  - The resource is processed last. If it succeeds, the other resources are told to commit; if it fails, they are told to roll back.

ORACLE

If you choose to use the Support Global Transactions option, which allows connections from the data source to participate in a global transaction even if you are not using an XA driver, you also need to select how this data source will participate in global transactions. Logging Last Resource (LLR) is one of the options. With it, this resource, within the transaction, is processed last and as a local transaction. If the database successfully commits locally, the remaining global transaction participants are told to commit. If the database locally rolls back, the remaining global transaction participants are told to roll back.

For more information, refer back to the lesson titled “Configuring JDBC.”

Logging Last Resource and Performance

- Even if XA drivers are available, you may want to configure your data source with a non-XA driver and select Logging Last Resource (LLR), if this data source represents the only database participant in your global transactions. This will improve performance and has the same ACID guarantee as XA.
- Non-XA drivers with LLR improves performance by:
  - Removing the need for an XA JDBC driver to connect to the database. XA JDBC drivers are typically less efficient than non-XA JDBC drivers.
  - Reducing the number of processing steps to complete the transaction, which also reduces network traffic and I/O.
  - Removing the need for XA processing at the database level (if the database is the only non-XA resource).

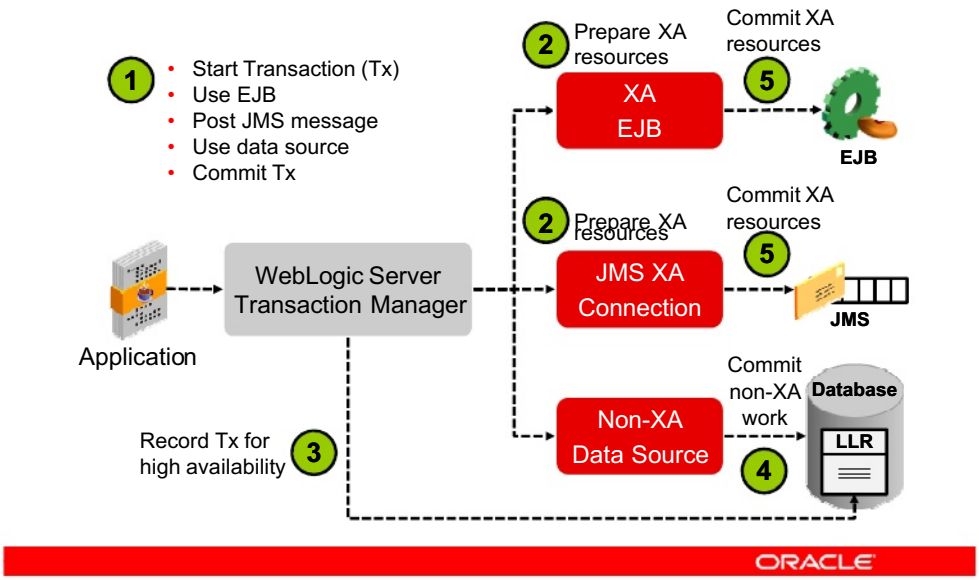
WebLogic Server supports the Logging Last Resource (LLR) transaction optimization through its data sources. LLR is a performance enhancement that enables one non-XA resource to participate in a global transaction with the same ACID guarantee as XA. The LLR resource uses a local transaction for its transaction work. The WebLogic Server transaction manager prepares all other resources in the transaction and then determines the commit decision for the global transaction based on the outcome of the LLR resource's local transaction.

The LLR optimization improves performance by:

- Using non-XA drivers rather than XA drivers. Non-XA drivers are generally more efficient.
- Reducing the number of processing steps to complete a transaction, which also reduces network traffic and the number of disk I/Os.
- Removing the need for XA processing at the database level.

Note that LLR improves performance for insert, delete, and update operations. However, for read operations with LLR, performance is somewhat slower than read operations with XA.

LLR: Example



WebLogic Server maintains an LLR table on the database to which a JDBC LLR data source pools its connections. This table is used for storing transaction log records, and is automatically created. If multiple LLR data sources are deployed on the same WebLogic Server instance and connect to the same database instance and schema, they will share the same LLR table. LLR table names are automatically generated unless administrators choose to configure them. The default table name is `WL_LLR_`*`SERVERNAME`*.

In a global transaction with an LLR participant, the WebLogic Server transaction manager follows these basic steps:

1. It receives a commit request from the application.
2. It calls a “prepare” on all XA-compliant transaction participants.
3. It inserts a commit record to the LLR table on the LLR participant (rather than to the usual transaction log).
4. It commits the LLR participan t’s local transaction (which includes both the transaction commit record insert and the application’s SQL work).
5. It calls a commit on all other transaction participants.
6. After the transaction completes successfully, it later deletes the databa se transaction log entry as part of a future transaction.

Transaction Log (TLog)

- During a transaction, the server writes to a binary transaction log (TLog).
  - The transaction log is not like other WebLogic Server logs. You do not view it (it is binary).
- If the server fails, when it is restarted, it reads its TLog to be able to recover transactions.
- If the server cannot be brought back up on the same machine due to a hardware failure, it can be started on a new hardware.

The TLog must be available.

Each server has a transaction log that records information about the propagation of a transaction through the system. The transaction log is written to persistent storage and assists the server in recovering from system crashes and network failures. You do not view a transaction log because it is in a binary format. WebLogic Server creates a new transaction log each `CheckpointIntervalSeconds`

To take advantage of the migration capability of the Transaction Recovery Service for servers in a cluster, you must store the transaction log in a location that is available to the server and its backup servers, in some shared storage location.

Configuring the Default Store

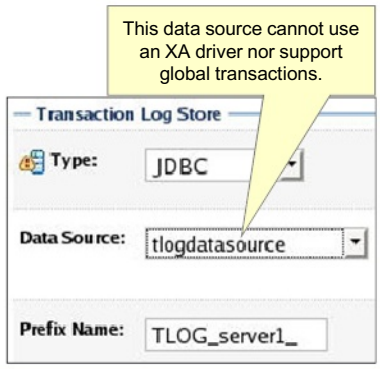
- The transaction log can be file based or in a database.
  - By default, the transaction log uses the default store.
    - The default store is file based.
    - If the default store directory is not set, it is here by default: `<domain>/servers/<server>/data/store/default`
- To change the default store directory:
  1. Select a server from the Servers table and then select **Configuration** > **Services**.
  2. Under Default Store, change the Directory field. For transaction recovery, change it to some reliable shared storage directory.

Each server has a default persistent store, which is a file-based store. By default, the transaction manager uses this to store transaction logs. To enable migration of the Transaction Recovery Service, you must change the configuration of the default store. For highest availability, use either a Storage Area Network (SAN) or other reliable shared storage. The use of NFS mounts is discouraged, but supported. Most NFS mounts are not transactionally safe by default, and, to ensure transactional correctness, need to be configured using your NFS vendor documentation in order to honor synchronous write requests.

Note that the file-based persistent store is not used exclusively by the WebLogic Server transaction manager. This store is also used for persistence by the diagnostic service, the JMS subsystem, and other WebLogic Server elements. For more information, see the section titled “Overview of the Persistent Store” in the *Configuring Server Environments for Oracle WebLogic Server* document.

Configuring a JDBC Transaction Log

- To use a database transaction log:
  1. Select a server from the Servers table and then select **Configuration** > **Services**.
  2. Under Transaction Log Store, use the Type drop-down list and select **JDBC**.
  3. Use the Data Source drop-down list to select a data source that has already been created and configured.
  4. Enter a Prefix Name for the table.



- **Type:** Select `JDBC` if you want transactions logged to a specific JDBC data source. (The default for Type is `Default Store`.)
- **Data Source:** The JDBC data source used by the transaction manager to store transaction logs.
  - Important notes about the data source: You cannot configure the transaction log store to use a JDBC data source that is configured to use an XA JDBC driver nor configured to support global transactions.
- **Prefix Name:** When using multiple TLOG JDBC stores, use this attribute to create a label ending in "_." This label is prepended to the name of the server hosting the JDBC TLOG store. After the server name, another "_" is appended to the string to form a unique JDBC TLOG store name. The default prefix name is "`TLOG_`" . For example, a valid JDBC TLOG store name, using the default Prefix Name, is `TLOG_myserver_` where `TLOG_` is the Prefix Name and `myserver` is the name of the server hosting the JDBC TLOG store.

Note: By default, the table used for the TLog is named `WLStore`. If it does not already exist, WebLogic Server creates it by using a default Data Definition Language (DDL) file. Under the Advanced section of the Transaction Log Store settings, you can choose your own DDL file (and other settings, as well).

Comparing File Store to JDBC Store

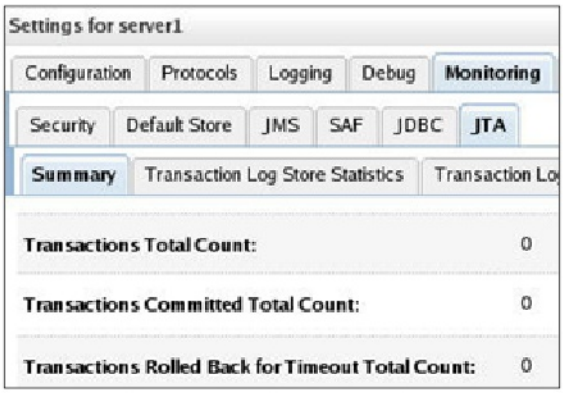
	FileStore	JDBCStore
Default store	It is the default	Cannot be used for the default
Transactions	Both have the same transaction guarantees and semantics	
Interface	The application interface is the same	
Throughput	Better	Worse
Configuration	Easier	Harder
Failure recovery	The file store must be configured to reside on shared storage	Made easier as all servers can use JDBC (data sources) to access the store
Backup and recovery	Shared storage another item to back up	Simplifies it, as database backup and recovery can include the TLog

You can configure a JDBC TLOG store to persist transaction logs to a database, which provides the following benefits:

- JDBC stores may make it easier to handle failure recovery because the JDBC interface can access the database from any machine on the same network.
- It can leverage the replication and high availability characteristics of the underlying database.
- It simplifies disaster recovery by allowing the easy synchronization of the state of the database and TLOGs.

Monitoring Transactions

- Select a server from the Servers table and then select **Monitoring** > **JTA**. Then select one of the many subtabs under **JTA**.



ORACLE

Under the **Summary** tab:

- **Transactions Total Count:** The total number of transactions processed. This total includes all committed, rolled back, and heuristic transaction completions since the server was started.
- **Transactions Committed Total Count:** The total number of transactions committed since the server was started
- **Transactions Rolled Back Total Count:** The number of transactions that were rolled back since the server was started
- **Transactions Rolled Back for Timeout Total Count:** The number of transactions that were rolled back due to a timeout
- **Transactions Rolled Back for Resource Errors Total Count:** The number of transactions that were rolled back due to a resource error
- **Transactions Rolled Back for Application Errors Total Count:** The number of transactions that were rolled back due to an application error
- **Transactions Rolled Back for System Errors Total Count:** The number of transactions that were rolled back due to an internal system error

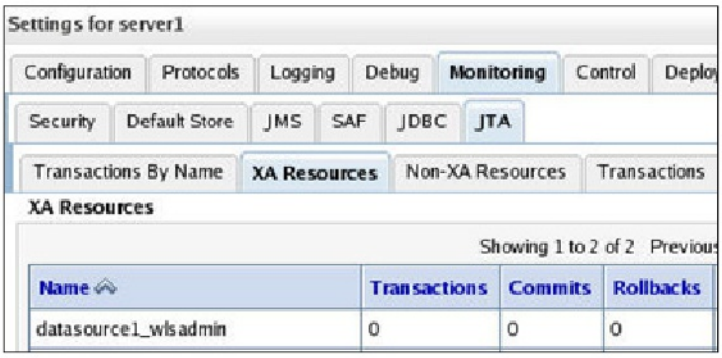
- **Heuristic Completions Total Count:** The number of transactions that completed with a heuristic status since the server was started
- **Abandoned Transactions Total Count:** The total number of transactions that were abandoned since the server was started
- **Transaction No Resources Committed Total Count:** The total number of transactions with no enlisted resources that were committed since the server was started
- **Transaction One Resource One Phase Committed Total Count:** The total number of transactions with only one enlisted resource that were one-phase committed since the server was started
- **Transaction Read Only One Phase Committed Total Count:** The total number of transactions with more than one enlisted resource that were one-phase committed due to read-only optimization since the server was started
- **Transaction Two Phase Committed Total Count:** The total number of transactions with more than one enlisted resource that were two-phase committed since the server was started
- **Transaction LLR Committed Total Count:** The total number of LLR transactions that were committed since the server was started
- **Active Transactions Total Count:** The number of active transactions on the server

Other JTA subtabs:

- **Transaction Log Store Statistics:** Runtime statistics for the transaction log store for this server displayed in a configurable table
- **Transaction Log Store Connections:** Runtime statistics for the active transaction log store connections displayed in a configurable table
- **Transactions By Name:** This page shows statistics about named transactions coordinated by the server.
- **XA Resources:** Use this page to monitor XA resource transactions coordinated by this server.
- **Non-XA Resources:** This page shows information about transactions in which non-XA resources on the server participate.
- **Transactions:** This page shows information about current transactions coordinated by the server or in which server resources participate. It also allows you to select a transaction that is in work and force a commit or rollback.
- **Recovery Services:** This page shows information about transactions that were processed by the server as part of recovery on server startup or after a crash.

Viewing Transaction Statistics for a Resource

To view transactional outcomes for a particular resource: Select the server, then **Monitoring** > **JTA** and either **XA Resources** or **Non-XA Resources**.



ORACLE

The columns in the table are:

- **Name:** The name of the XA resource that participated in the global transactions
- **Transactions:** The total number of transactions processed. This total includes all committed, rolled back, and heuristic transaction completions since the server was started.
- **Commits:** The number of transactions that were committed since the server was started
- **Rollbacks:** The number of transactions that were rolled back since the server was started
- **Timeout Rollbacks:** The number of transactions that were rolled back due to a timeout
- **Resource Rollbacks:** The number of transactions that were rolled back due to a resource error
- **Application Rollbacks:** The number of transactions that were rolled back due to an application error
- **System Rollbacks:** The number of transactions that were rolled back due to an internal system error

- **Heuristics:** The number of transactions that completed with a heuristic status since the server was started
- **Transaction Abandoned Total Count:** The total number of transactions that were abandoned since the server was started

Notes:

- If you noticed that the subtabs under **Monitoring** > **JTA** look different here than in a previous slide, it is because not all tabs are shown in this screenshot due to space limitations.
- The **Non-XA Resources** tab gives information about transactions in which non-XA resources participated.
- A heuristic status (or heuristic decision) is a decision made by one or more resources in a transaction to commit or roll back without first getting the consensus outcome that is determined by the resource manager. A resource typically makes a heuristic decision only under abnormal circumstances, such as a communication failure. When a resource makes a heuristic decision, there is a chance that the decision made differs from that of the transaction manager, resulting in a loss of data integrity.

Forcing a Commit or Rollback

Under a server's **Monitoring** > **JTA** > **Transactions** tab, current transactions are listed, with information about them, including their status.

- If a transaction is "stuck," due to some system or network failure, eventually the Abandon Timeout period will elapse and the transaction will be removed (with a heuristic error written to the server log). Before then, the transaction can be selected and a button pressed to force the transaction to completion. The buttons are:
  - Force Local Commit
  - Force Global Commit
  - Force Local Rollback
  - Force Global Rollback

The server's **Monitoring** > **JTA** > **Transactions** page shows information about current transactions coordinated by the server or in which server resources participate. This page allows you to select a transaction that is in-process and force a commit or rollback. The buttons are:

- **Force Local Commit:** Each participating resource is issued a commit operation for the selected transaction. If the local server is the coordinator for the transaction, the "commit record" is released.
- **Force Global Commit:** A local commit operation is attempted at each participating server for the selected transaction. If this option is invoked on a non-coordinating server, the coordinator will be contacted to process the operation. The coordinating server will issue asynchronous requests to each participant server.
- **Force Local Rollback:** Each participating resource is issued a rollback operation for the selected transaction.
- **Force Global Rollback:** A local rollback operation is attempted at each participating server for the selected transaction. If this option is invoked on a non-coordinating server, the coordinator will be contacted to process the operation. The coordinating server issues asynchronous requests to each participant server.

Forcing a Commit or Rollback

Transaction Status	Can Use Force Commit?	Can Use Force Rollback?
Active		Yes
Preparing		Yes
Prepared	Yes	Yes
Committing	Yes	
Committed	Yes	
Rolling Back		Yes
Rolled Back		Yes
Marked Roll Back		Yes
Unknown	Yes	Yes

What the status values mean:

- **Active:** The application is processing the transaction. The transaction has not yet reached the two-phase commit processing.
- **Preparing:** In the first phase of 2PC before all participants have responded: "ready to commit"
- **Prepared:** In between when all participants have responded to "prepare" but before the commit point or the initiation of rollback processing.
- **Committing:** The time from when the commit decision is made up to the point when all participants have been informed of the outcome and the commit is complete.
- **Committed:** The transaction has been committed. It is likely that heuristics exist, otherwise the transaction would have been completed and would not have been displayed in the list of current transactions.
- **Rolling Back:** This state occurs from the point when rollback processing is initiated up to the point when all participants have been instructed to roll back and the rollback is complete.

- **Rolled Back:** The transaction has been rolled back. It is likely that heuristics exist, otherwise the transaction would have been destroyed and would not have been displayed in the list of current transactions.
- **Marked Roll Back:** The transaction has been marked for rollback, perhaps as a result of a `setRollbackOnly()` method being called in code.
- **Unknown:** The current status cannot be determined.

Troubleshooting Transactions

- Use the monitoring capabilities of the administration console.
  - With all the possible states of a transaction, you can see how far along in the process an active transaction is.
- Use the server logs.
  - If a message is logged during a transaction, the transaction ID is part of that log message.
  - Look for exceptions in the logs. WebLogic JTA supports all the standard JTA exceptions.
    - It extends the `RollbackException` class to preserve the srcinal reason for the rollback.
  - Turn on transaction debug flags for more detailed log messages.

JTA Debugging Scopes (those not currently used by WebLogic Server are not listed):

- `DebugJTAXA` (scope `weblogic.transaction.xa`) – Traces for XA resources
- `DebugJTANonXA` (scope `weblogic.transaction.nonxa`) – Traces for non-XA resources
- `DebugJTAXAStackTrace` (scope `weblogic.transaction.stacktrace`) – Detailed tracing that prints stack traces at various critical points
- `DebugJTA2PC` (scope `weblogic.transaction.twopc`) – Traces all two-phase commit operations
- `DebugJTA2PCStackTrace` (scope `weblogic.transaction.twopcstacktrace`) – Detailed two-phase commit tracing that prints stack traces
- `DebugJTATLOG` (scope `weblogic.transaction.tlog`) – Traces transaction logging information
- `DebugJTAJDBC` (scope `weblogic.transaction.jdbc`, `weblogic.jdbc.transaction`) – Traces information about reading and writing JTA records

- `DebugJTARecovery` (scope `weblogic.transaction.recovery`) – Traces recovery information
- `DebugJTAGateway` (scope `weblogic.transaction.gateway`) – Traces information about imported transactions. The WebLogic Server transaction manager exposes a interface that can be used to import transactions from a foreign transaction manager. The WebLogic Server transaction manager then acts as the coordinator for the imported transactions within WebLogic Server.
- `DebugJTAGatewayStackTrace` (scope `weblogic.transaction.gatewaystacktrace`) – Stack traces related to imported transactions
- `DebugJTANaming` (scope `weblogic.transaction.naming`) – Traces transaction naming information
- `DebugJTANamingStackTrace` (scope `weblogic.transaction.namingstacktrace`) – Traces transaction naming information
- `DebugJTAResourceHealth` (scope `weblogic.transaction.resourcehealth`) – Traces information about XA transaction resource health
- `DebugJTAMigration` (scope `weblogic.transaction.migration`) – Traces information about Transaction Log migration
- `DebugJTALifecycle` (scope `weblogic.transaction.lifecycle`) – Traces information about the transaction server lifecycle (initialization, suspension, resuming, and shutdown)
- `DebugJTALLR` (scope `weblogic.transaction.llr`) – Traces all Logging Last Resource operations
- `DebugJTAHealth` (scope `weblogic.transaction.health`) – Traces information about transaction subsystem health
- `DebugJTATransactionName` (scope `weblogic.transaction.name`) – Traces transaction names
- `DebugJTAResourceName` (scope `weblogic.transaction.resourcename`) – Traces transaction resource names

Quiz

A global (distributed) transaction involves more than one _____.

a. WebLogic Server
b. Cluster
c. Transactional resource
d. Domain
e. Continent

Answer: c

Quiz

JTA stands for:

a. Just in Time Architecture
b. Java Transaction Architecture
c. Job Transaction API
d. Java Transaction API

Answer: d

Summary

In this lesson, you should have learned how to:

- Describe WebLogic Server's role in managing transactions
- Configure a database persistent store for transactions

Practice 15-1 Overview: Configuring Transaction Persistence

This practice covers configuring a database as the persistent store for transaction logs.

16

WebLogic Server Security

Objectives

After completing this lesson, you should be able to:

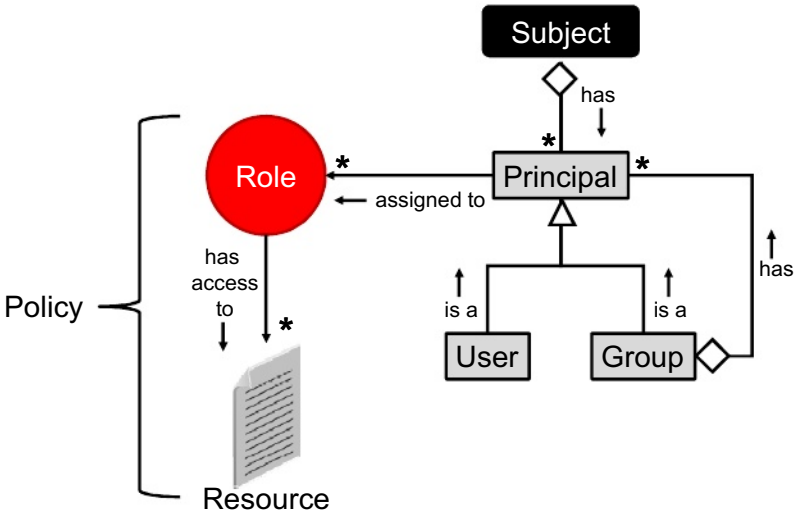
- Describe the basics of the WebLogic Server security architecture
- Describe basic LDAP concepts
- Configure an external LDAP authentication provider for WebLogic Server

Some Security Terms

- *Subject*: The user (or service) accessing the system
  - A subject has one (or more) *principals*
- *Principal*: The unique identity of a subject, assigned after authentication
  - Usually a username or a group name
- *User*: An individual (or program) accessing the application
- *Credentials*: Usually username or password
- *Group*: A collection of users and/or other groups
- *Role*: A type of user
  - Principals can be assigned roles to say what kind of user they represent
- *Policy*: A security rule, usually an association of a resource to one or more roles

An example of a policy that is not an association of a resource to a role would be a “daytime access” rule: This particular resource (or set of resources) may only be accessed between the hours of 8:00 AM and 5:30 PM.

Some Security Terms: Graphically



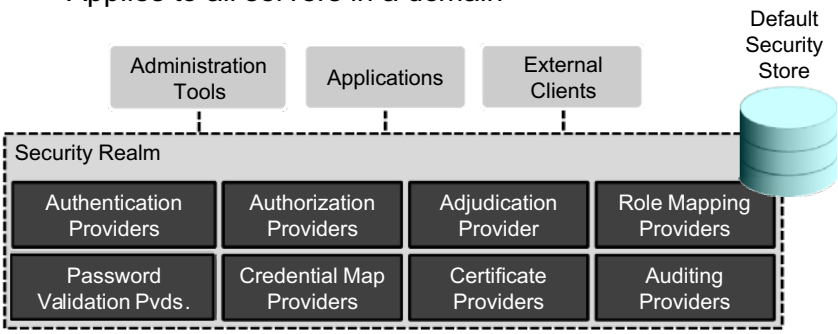
A User is a Principal. A Group is a Principal. A Subject, after authentication, is assigned one or more Principals (before authentication the Subject has zero Principals). A Group contains zero or more Principals (Users and other Groups). A Principal (a User or, more often, a Group) is assigned to zero or more Roles. A Policy states that a particular Role has access to a particular Resource (or set of Resources).

Note: The "*" means "zero or more."

WebLogic Server Security Realm

A WebLogic Server security realm:

- Handles security logic and decisions for a domain
- Consists of a series of pluggable security providers
- Applies to all servers in a domain



ORACLE

The security service in WebLogic Server simplifies the configuration and management of security while offering robust capabilities for securing WebLogic Server. Security realms act as a scoping mechanism. Each security realm consists of a set of configured security providers, users, groups, security roles, and security policies. You can configure multiple security realms in a domain; however, only one security realm can be active.

A security policy is an association between a WebLogic resource and one or more security roles. Security policies protect the WebLogic resource against unauthorized access. A WebLogic resource has no protection until you create a security policy for it.

A security provider store contains the users, groups, security roles, security policies, and credentials used by some types of security providers to provide their services. For example, an authentication provider requires information about users and groups; an authorization provider requires information about security policies; a role mapping provider requires information about security roles, and a credential mapping provider requires information about credentials to be used to access remote applications. These security providers need this information to be available to function properly.

What the Providers Do

- **Authentication:** Who are you? Prove it.
  - Can optionally use an Identity Assertion Provider, which takes a token from outside of WebLogic Server, validates it, and, if valid, maps the token to a username.
- **Authorization:** Are you allowed to use this resource?
  - Uses the Role Mapping provider
- **Adjudication:** The multiple authorization providers do not agree. Can the user have the resource?
- **Role Mapping:** What type of user are you?
  - For example: `manager`, `salesperson`, `administrator`
- **Password Validation:** Does the new or modified password meet the password rules?

- **Authentication:** The authentication provider determines whether a user is legitimate. When a user logs in with a username and password, this provider validates that the username exists and that the password entered is correct. After successfully proving an identity, an authentication context is established (a subject containing one or more principals), which allows an identified user or system to be authenticated to other entities.
- **Identity Assertion:** Identity Assertion providers are used as part of a perimeter authentication process. When perimeter authentication is used, a token from outside of the WebLogic Server domain is passed to an active identity assertion provider in a security realm that is responsible for validating tokens of that type. If the token is successfully validated, the identity assertion provider maps the token to a WebLogic Server username, and sends that username back to WebLogic Server, which then continues the authentication process.

- **Authorization:** The authorization process is initiated when a user or system requests a WebLogic Server resource on which it will attempt to perform a given operation. The resource container that handles the type of resource receives the request (for example, the EJB container receives the request for an EJB resource). The resource container calls the WebLogic Security Framework and passes in the request parameters, including information such as the subject of the request and the WebLogic Server resource being requested. The WebLogic Security Framework calls the role mapping provider and passes in the request parameters in a format that the role mapping provider can use. The role mapping provider uses the request parameters to compute a list of roles to which the subject making the request is entitled, and passes the list of applicable roles back to the WebLogic Security Framework. The authorization provider determines whether the subject is entitled to perform the requested action on the WebLogic Server resource.
- **Adjudication:** The adjudication provider determines what to do if multiple authorization providers' access decisions do not agree. The adjudication provider resolves authorization conflicts by weighing each access decision and returning a final result.
- **Role Mapping:** The WebLogic Security Framework calls each role mapping provider that is configured as part of an authorization decision (see the explanation of the authorization provider). The role mapping provider returns the list of roles a particular user has. These roles are returned to the WebLogic Security Framework, where they can be used in an access decision. WebLogic Server resources can be configured so that certain roles can perform certain actions. (For example, in a web application, resources (given as URL patterns), can be protected so that only a user with the proper role is allowed access to them.)
- **Password Validation:** When the password validation provider is configured with an authentication provider, the authentication provider invokes the password validation provider whenever a password is created or updated. The password validation provider then performs a check to determine whether the password meets the criteria established by a set of configurable rules.

What the Providers Do

- **Credential Mapping:** Maps a user authenticated to WebLogic Server to a set of credentials for another system, so that the user can access that other system
- **Certificate Providers:** Keeps a list of trusted digital certificates and validates those certificates
- **Auditing:** For certain user tasks, tracks who did what and when

- **Credential Mapping:** The credential mapping provider is used to associate a WebLogic Server user to their credentials for some other system, to be used with a Resource Adapter. The provider maps a user's authentication credentials (username and password) to those required for some legacy application, so that the legacy application receives the necessary credential information and allows access.
- **Certificate Providers:** The certificate lookup and validation providers complete certificate chains and validate them. If multiple providers are configured, a certificate or certificate chain must pass validation with all of them in order for the certificate or certificate chain to be accepted. (A certificate chain, also known as the certification path, is a list of certificates used to authenticate an entity. The chain begins with the certificate of that entity, and each certificate in the chain is signed by the entity identified by the next certificate in the chain. The chain terminates with a root Certificate Authority (CA) certificate and is signed by the CA.)
- **Auditing:** An auditing provider collects, stores, and distributes information about operating requests and the outcome of those requests for the purposes of non-repudiation (users cannot later say that they did not perform some task). An auditing provider makes the decision about whether to audit a particular event based on specific audit criteria, including audit severity levels.

Security Stores

A persistent store is assigned to a security realm to persist assets such as:

- Users and groups
- Roles
- Policies
- Credential maps
- Certificates

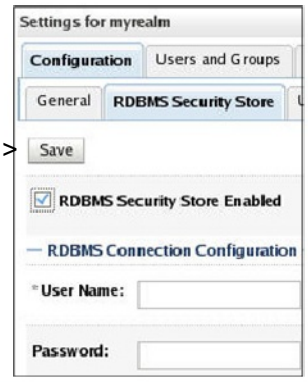
Some providers use the default security store while others use an external system.

The default Auditing and Adjudication providers do not use the persistent security stores configured for their parent security realm.

If you have multiple security providers of the same type configured in the same security realm, these security providers may use the same security provider database. For example, if you configure two WebLogic authentication providers in the default security realm (called `myrealm`), both WebLogic authentication providers would use the same location in the embedded LDAP server as their security provider database and thus will use the same users and groups. Furthermore, if you or an administrator add a user or group to one of the WebLogic authentication providers, you will see that user or group appear for the other WebLogic authentication provider as well.

Default Security Store Implementation

- The WebLogic default:
  - An embedded LDAP server running on the admin server and replicated to the managed servers
- Or, you can configure the RDBMS security store:
  1. In the admin console, select the realm. Then select **Configuration** > **RDBMS Security Store**.
  2. Select **RDBMS Security Store Enabled** and fill in the required fields.
     - The schema files are located at `<WL_HOME>/server/lib`.



ORACLE

WebLogic Server uses its embedded LDAP server to store users, groups, security roles, and security policies for the WebLogic security providers. The embedded LDAP server is a complete LDAP server that is production quality for reasonably small environments (10,000 or fewer users). For applications that need to scale above this number, the embedded LDAP server can serve in the development and integration environments for future export to an external LDAP server for the test and production environments.

When the RDBMS security store is configured in a security realm, any of the following security providers that have been created in the security realm automatically uses only the RDBMS security store, and not the embedded LDAP server:

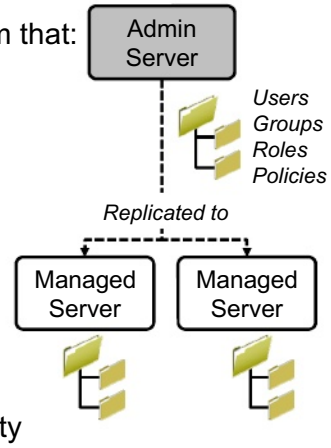
- XACML Role Mapping and Authorization
- Default, PKI, and SAML Credential Mapping
- SAML Identity Assertion

Other security providers continue to use their default security stores; for example, the WebLogic authentication provider continues to use the embedded LDAP server. Note that the use of the RDBMS security store is required to use SAML 2.0 services in two or more WebLogic Server instances in a domain, such as in a cluster.

Default Security Configuration

A new domain includes a default realm that:

- Includes default providers:
  - Default authenticator
  - Default identity asserter
  - XACML* role mapper
  - XACML* authorization provider
  - Default password validator
  - Default credential mapper
  - Default certificate path provider
    - Validates certificate chains
- Uses the embedded LDAP security store



* eXtensible Access Control Markup Language: An XML-based security policy language

ORACLE

To simplify the configuration and management of security, WebLogic Server provides a default security configuration. In the default security configuration, a default security realm, `myrealm`, is set as the active security realm, and the WebLogic authentication, identity assertion, credential mapping, certification path, password validation, EXtensible Access Control Markup Language (XACML) authorization, and XACML role mapping providers are defined as the security providers in this security realm. (XACML is OASIS standard, XML-based, security policy, and access control language.)

The WebLogic Server embedded LDAP server for a domain consists of a master LDAP server, maintained in the domain's administration server, and a replicated LDAP server maintained in each managed server in the domain. When changes are made using a managed server, updates are sent to the embedded LDAP server on the administration server. The embedded LDAP server on the administration server maintains a log of all changes. The embedded LDAP server on the administration server also maintains a list of managed servers and the current change status for each one. The embedded LDAP server on the administration server sends appropriate changes to each managed server. This process occurs when an update is made to the embedded LDAP server on the administration server. However, depending on the number of updates, it may take several seconds or more for the changes to be replicated to the managed servers.

Security Customization Approaches

- Create an entirely new security realm and add (at least) the required providers.
  - After the new security realm is configured, make it the active security realm.
- Add, remove, and configure providers in the default realm, called `myrealm`.
- Have developers create custom security providers and add them to either the default realm or a custom security realm.

Most realm modifications require a domain restart.

ORACLE

The easiest way to customize the default security configuration is to add the security providers you want to the default security realm (`myrealm`). Many customers instead create an entirely new security realm and place in it the security providers they want. This preserves your ability to revert more easily to the default security configuration. You configure security providers for the new realm, migrate any security data, such as users and groups, from the existing default realm, and then set the new security realm as the default realm.

A valid security realm requires an authentication provider, an authorization provider, an adjudication provider, a credential mapping provider, a role mapping provider, and a certification path builder. Optionally, define identity assertion, auditing, and certificate registry providers. If you configured the default authentication, authorization, credential mapping, or role mapping provider or the default certificate registry in the new security realm, verify that the settings of the embedded LDAP server are appropriate.

Authentication Providers

Authentication providers are organized into two categories:

- *Authenticators:*
  - Establish the user's identity given some credentials (like username and password)
  - Can associate multiple principals with a single user, such as groups
- *Identity asserters:*
  - Validate tokens claiming a user has already been authenticated
  - Allow WebLogic Server to participate in single sign-on (SSO) solutions
  - Can map the token to a local user and use authenticators to look up that user's principals

Authentication providers are used to prove the identity of users or system processes. Authentication providers also remember, transport, and make that identity information available to various components of a system (via subjects) when needed.

Both users and groups can be used as principals by application servers like WebLogic Server. A principal is an identity assigned to a user or group as a result of authentication. The Java Authentication and Authorization Service (JAAS) requires that subjects be used as containers for authentication information, including principals. Each principal stored in the same subject represents a separate aspect of the same user's identity, much like credit cards in a person's wallet.

An *identity assertion provider* is a specific form of authentication provider that allows users or system processes to assert their identity using tokens supplied by clients. Typical token types include X509 certificates, SAML, and Kerberos. Identity assertion providers enable perimeter authentication and support single sign-on (SSO). For example, an identity assertion provider can generate a token from a digital certificate, and that token can be passed around the system so that users are not asked to sign on more than once.

Unlike in a simple authentication situation, identity assertion providers do not verify credentials such as usernames and passwords. They verify that the user exists.

Available Authentication Providers

- Available authenticators include:
  - Default (Internal LDAP)
  - LDAP (generic and vendor-specific)
  - Database (multiple DBMS providers)
  - Windows NT
  - SAML (Security Assertion Markup Language)
- Available identity asserters include:
  - Default
  - LDAP X509
  - SAML
  - Negotiate (SPNEGO)

The default WebLogic Server authentication provider manages users and groups in the embedded LDAP server.

LDAP authentication providers access external LDAP stores. WebLogic Server provides LDAP authentication providers that access Oracle Internet Directory, Oracle Virtual Directory, Open LDAP, iPlanet, Microsoft Active Directory, and Novell Directory Service stores.

A DBMS authentication provider is a username and password authentication provider that uses a relational database (rather than an LDAP server) as its data store for user, password, and group information.

The SAML authentication provider may be used with the SAML 1.1 or SAML 2.0 identity assertion provider to allow virtual users to log in via SAML. This provider creates an authenticated subject using the username and groups retrieved from a SAML assertion. (SAML is the Security Assertion Markup Language. It is an XML-based, OASIS standard data format for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider.)

The Windows NT authentication provider uses account information defined for a Windows NT domain to authenticate users and groups and to permit Windows NT users and groups to be listed in the administration console.

The default identity assertion provider supports identity assertion with X509 certificates and CORBA Common Secure Interoperability version 2 (CSI v2). You can also map the tokens authenticated by the identity assertion provider to users in the security realm.

The LDAP X509 identity assertion provider receives an X509 certificate, looks up the LDAP object for the user associated with that certificate, ensures that the certificate in the LDAP object matches the presented certificate, and then retrieves the name of the user from the LDAP object.

The SAML identity assertion provider acts as a consumer of SAML security assertions, allowing WebLogic Server to participate in SSO solutions for web or web service applications. It validates assertions by checking the signature and validating the certificate for trust based on data configured for the associated partner. The provider then extracts the identity information contained in the assertion, and maps it to a local subject in the security realm.

The Negotiate identity assertion provider enables SSO with Microsoft clients. The provider decodes Simple and Protected Negotiate (SPNEGO) tokens to obtain Kerberos tokens, validates the Kerberos tokens, and maps Kerberos tokens to WebLogic users.

Lightweight Directory Access Protocol (LDAP)

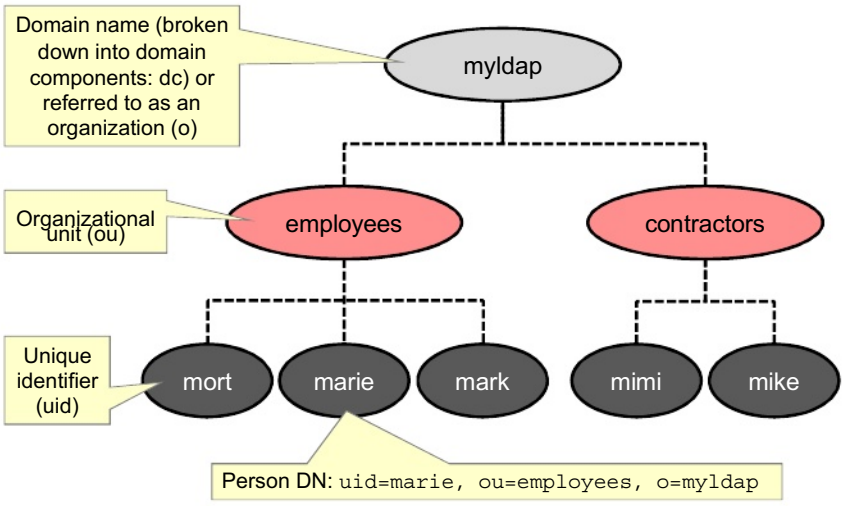
- LDAP:
  - Is a TCP/IP protocol
  - Provides a hierarchical lookup and search service
  - Models information as a tree of entries, whose attributes are defined by a schema or “object class”
  - Defines default schemas for common entries like people and groups
  - Supports SSL
- Entries:
  - Identify their locations in the tree by using a *distinguished name* (DN)
  - Can be referrals that link to other LDAP servers

The Lightweight Directory Access Protocol, better known as LDAP, is a protocol that provides access to a compliant directory of information via TCP/IP (Transmission Control Protocol/Internet Protocol). The strengths of LDAP-compliant directories include speed, simplicity, and the ability to be replicated and distributed across several servers. An LDAP directory can be used to store a great deal of information from user login credentials to company telephone directories.

Unlike databases that are designed for processing hundreds or thousands of changes per minute, LDAP directories are heavily optimized for read performance. LDAP is intentionally designed for environments where search operations are much more common than modify operations.

LDAP Version 3 implements a referral mechanism that allows servers to return references to other servers as a result of a directory query. This makes it possible to distribute directories globally by partitioning a directory information tree (DIT) across multiple LDAP servers.

LDAP Structure



Directories are viewed as a tree, analogous to a computer's file system. Each entry in a directory is called an object. These objects are of two types: containers and leaves. A container is like a folder; it contains other containers or leaves. A leaf is simply an object at the end of a branch. A tree cannot contain any arbitrary set of containers and leaves. It must match the schema defined for the directory.

The top level of the LDAP directory tree is the base, referred to as the base DN. A base DN can be one of several forms. Here are some examples:

- A domain name, broken into components (`dc=Acme,dc=com`)
- An organization name (`o=Acme Corp`)
- An organization name along with a country (`o=Acme Corp,c=India`)

Organizational units are standard LDAP object classes that act as containers for other entries. The identifying attribute for an organizational unit is "`ou`." The standard LDAP schema also defines a `person` class and a `group` class, which is a collection of people.

The `person` type also includes other attributes such as Common Name (a person's full name: `cn`), Unique Identifier (`uid`), Surname (last name: `sn`), and Password (`userpassword`).

LDAP Search Operations

Searching for LDAP entries involves:

1. The base DN from which to start searching
2. A search filter that specifies the:
   - Search criteria in terms of attribute values
   - The type or “object class” of the desired entries
3. An indication whether or not the search should include any child entries

LDAP Query Basics

- = (equal)
  - Example: `(uid=tjp)`
- & (logical and)
  - Example: `(&(uid=tjp)(sn=Parker))`
- | (logical or)
  - Example: `(|(uid=tjpark)(uid=tjp))`
- ! (logical not)
  - Example: `(!(sn=Parker))`
- * (wildcard)

Here is an LDAP search filter that finds all person entries whose user ID begins with "t," while ignoring those whose surname starts with "Th":

```
(&(&(uid=t*)(!(sn=Th*)))(objectclass=person))
```


The "&" represents a logical "and" when combining multiple expressions that have been grouped together in parentheses. Similarly, the "|" represents a logical "or," and a "!" represents a logical "not."

Examples:

- `(uid=tjp)` – All entries whose unique identifier is equal to `tjp`
- `(&(uid=tjp)(sn=Parker))` – All entries whose unique identifier is equal to `tjp` and whose surname is equal to `Parker`
- `(|(uid=tjpark)(uid=tjp))` – All entries whose unique identifier is equal to `tjpark` or equal to `tjp`
- `(!(sn=Parker))` – All entries whose surname is not equal to `Parker`

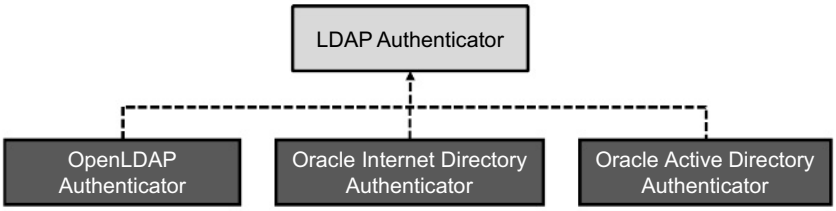
Search filters can specify one or more object classes. Here is an example:

```
(&(&(objectClass=person)(objectClass=organizationalPerson))
                                                (objectClass=user))
```

LDAP Authentication Providers

WebLogic Server includes:

- A base LDAP authenticator that can be configured to support any compliant vendor
- Vendor-specific LDAP authenticators, whose attributes are set to vendor-specific defaults for convenience



ORACLE

Each LDAP authentication provider stores user and group information in an external LDAP server. The providers differ primarily in how they are configured by default to match typical directory schemas for their corresponding LDAP server. For example, the generic authenticator is configured to use a person's common name (`cn`) as a user ID, while by default Oracle Internet Directory uses the "`uid`" attribute for this purpose. Similarly, the names of object classes used to represent people or groups may vary from vendor to vendor. For example, the generic authenticator is configured to use the object class "`groupofuniquenames`," while by default Oracle Internet Directory uses the object class "`groupofnames`."

WebLogic Server does not support or certify any particular LDAP servers. Any LDAP v2 or v3 compliant LDAP server should work with WebLogic Server.

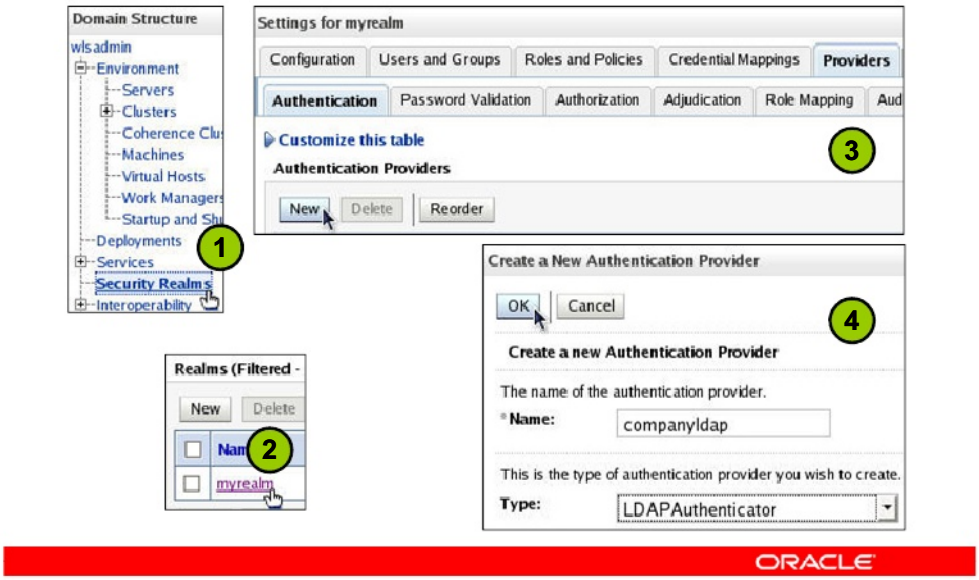
If an LDAP authentication provider is the only configured authentication provider for a security realm, you *must* include the `Admin` role and assign that role to a user or group of users in the LDAP directory. If the LDAP user who boots WebLogic Server is not properly added to a group that is assigned to the `Admin` role, and the LDAP authentication provider is the only authentication provider with which the security realm is configured, WebLogic Server cannot be booted.

Available LDAP Authentication Providers

- The available LDAP authentication providers include:
  - LDAP Authenticator (generic)
  - Oracle Internet Directory Authenticator
  - Oracle Virtual Directory Authenticator
  - iPlanet Authenticator
  - Active Directory Authenticator
  - Novell Authenticator
  - OpenLDAP Authenticator
- These providers:
  - Can be used to change passwords of existing users
  - Cannot be used to create, update, or delete users and groups

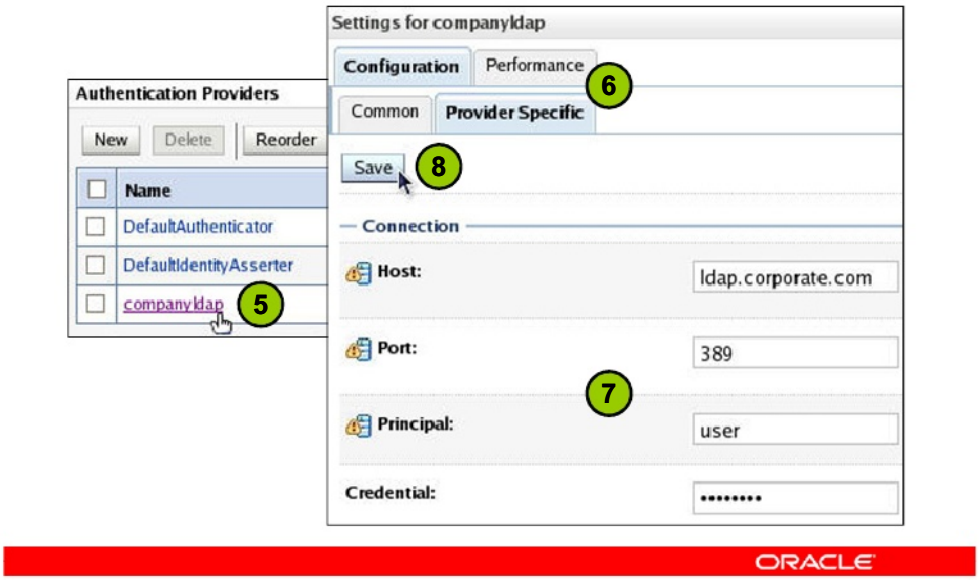
The LDAP authentication providers in WebLogic Server are configured to work readily with the Oracle Internet Directory, Oracle Virtual Directory, iPlanet, Active Directory, Open LDAP, and Novell NDS LDAP servers. You can use an LDAP authentication provider to access other types of LDAP servers. Choose either the generic LDAP authenticator or an existing LDAP provider that most closely matches the new LDAP server and customize the existing configuration to match the directory schema and other attributes for your LDAP server.

Creating a New LDAP Authentication Provider



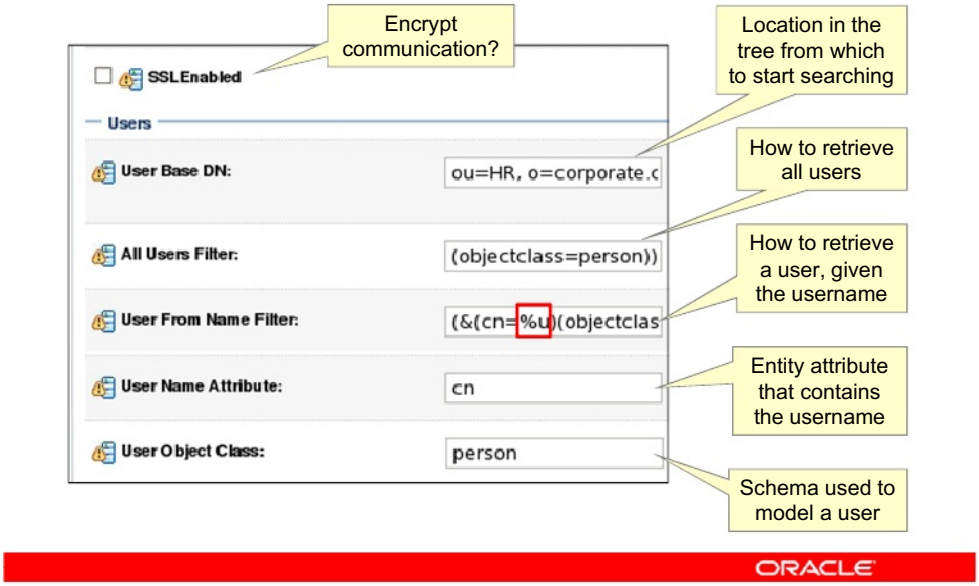
1. After locking the configuration, in the Domain Structure, select **Security Realms**.
2. In the Realms table, click the realm name.
3. Select **Providers** > **Authentication**. Click the **New** button.
4. Enter the **Name** of the provider and select the **Type** from the drop-down list. Then click **OK**.

Configuring the LDAP Provider: Connection



5. Click the name of the new provider in the Authentication Providers table.
6. Click **Configuration** > **Provider Specific**.
7. Enter values for the following connection attributes:
   - **Port:** The port of the LDAP server
   - **Principal:** The Distinguished Name (DN) of the LDAP user that WebLogic Server should use to connect to the LDAP server
   - **Credential:** The credential (usually a password) used to connect to the LDAP server
   - **SSL Enabled** (shown on next slide): Specifies whether the SSL protocol should be used when connecting to the LDAP server. For a more secure deployment, Oracle recommends using the SSL protocol to protect communication between the LDAP server and WebLogic Server.
8. Click **Save**. Then activate the changes.

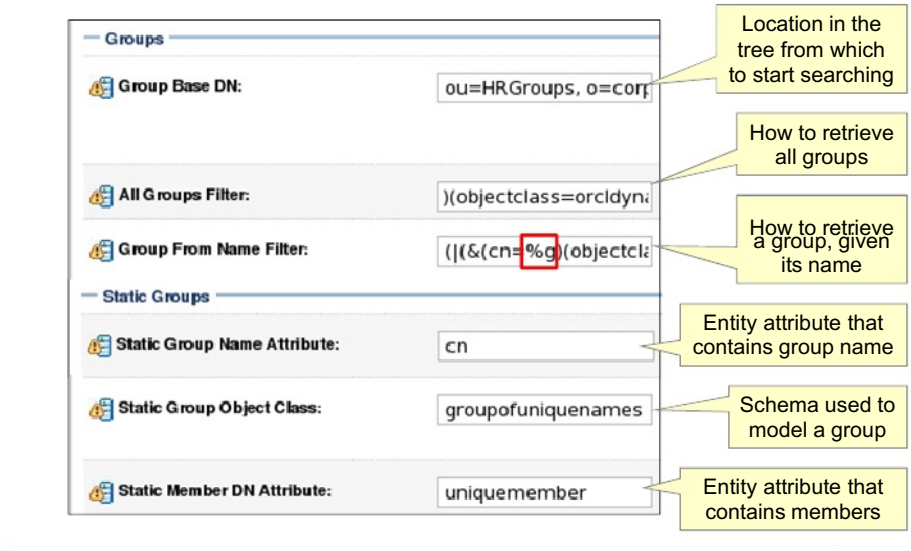
Configuring the LDAP Provider: Users



Enter values for any of the following user search attributes:

- **User Base DN:** The base distinguished name (DN) of the tree in the LDAP directory that contains users
- **All Users Filter:** The LDAP filter expression used to retrieve all users. If not specified, a simple default filter is generated based on the user object class.
- **User From Name Filter:** The LDAP filter expressions used to locate a user entry given its user ID. Use the token "`%u`" to indicate where the provider should insert the user ID before executing the search.
- **User Search Scope** (not shown): Specifies how deep in the LDAP directory tree the provider should search for users. Valid values are "`subtree`" and "`onelevel`."
- **User Name Attribute:** The attribute of an LDAP user object that specifies the user's login ID
- **User Object Class:** The LDAP object class that stores users
- **Use Retrieved User Name as Principal** (not shown): Specifies whether or not the username retrieved from the LDAP server should be used as the principal

Configuring the LDAP Provider: Groups



Enter values for any of the following group search attributes:

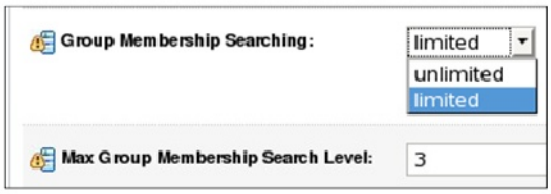
- **Group Base DN:** The base distinguished name (DN) of the tree in the LDAP directory that contains group definitions.
- **All Groups Filter:** An LDAP filter expression for finding all groups beneath the base group distinguished name (DN). If not specified, a simple default search filter is created based on the group object class.
- **Group From Name Filter:** An LDAP filter expression for finding a group given the name of the group. Use the "`%g`" token to indicate where the provider should insert the user ID before executing the search. If not specified, a simple default search filter is created based on the group schema.
- **Group Search Scope** (not shown): Specifies how deep in the LDAP directory tree to search for groups. Valid values are "`subtree`" and "`onelevel`."
- **Ignore Duplicate Membership** (not shown): Determines whether duplicate members are ignored when adding groups.

- **Static Group Name Attribute:** The attribute of a group object that specifies the name of the group
- **Static Group Object Class:** The name of the LDAP object class that stores groups
- **Static Member DN Attribute:** The attribute of a group object that specifies the distinguished names (DNs) of the members of the group
- **Static Group DNs from Member DN Filter** (not shown): Given the DN of a member of a group, returns the DNs of the groups that contain that member

Note: A static group contains a list of members that you explicitly administer. A dynamic group is one whose membership, rather than being maintained in a list, is computed, based on rules and assertions you specify.

Configuring the LDAP Provider: Subgroups

- Groups can include other groups.
- To improve performance, you can limit the depth that the provider will search for subgroups.

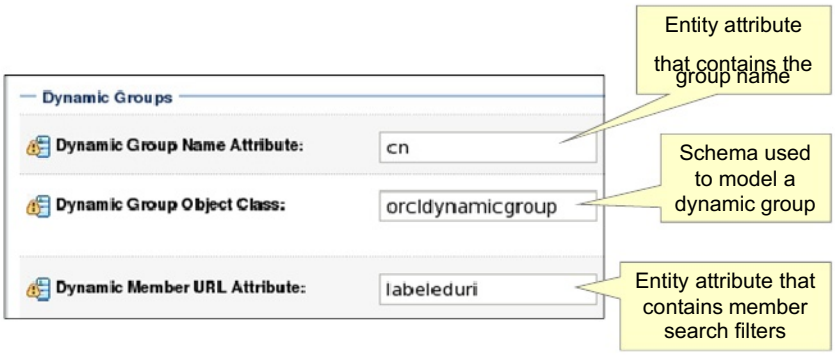


Enter values for any of the following attributes that apply to subgroup searching:

- **Group Membership Searching:** Specifies whether recursive group searches into nested groups are `unlimited` or `limited`. For configurations that use only the first level of nested group hierarchy, this attribute allows improved performance during user searches by limiting the search to the first level of the group.
- **Max Group Membership Search Level:** Specifies how many levels of group membership can be searched. This setting is valid only if Group Membership Searching is set to "`limited`." A value of `0` indicates that only direct groups will be found. That is, when searching for membership in Group A, only direct members of Group A will be found. If Group B is a member of Group A, the members of Group B will not be found by this search. Any non-zero value indicates the number of levels to search. For example, if this attribute is set to `1`, a search for membership in Group A will return direct members of Group A. If Group B is a member of Group A, the members of Group B will also be found by this search. However, if Group C is a member of Group B, the members of Group C will not be found by this search.

Configuring the LDAP Provider: Dynamic Groups

- Instead of a list of users, dynamic groups contain a list of search filters, each of which returns zero or more users.
- Member search filters are expressed as URLs.



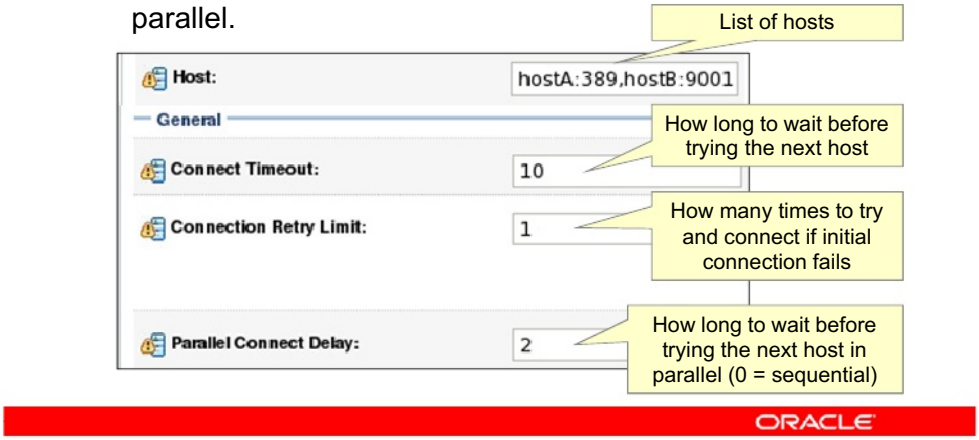
Many LDAP servers have a concept of dynamic groups or virtual groups. These are groups that, rather than consisting of a list of users and groups, contain some queries or code that define the set of users. The term “dynamic” describes the means of defining the group and not any runtime semantics of the group within WebLogic Server.

The provider attributes for dynamic groups are very similar to static ones, but the following additional attributes are also available:

- **Dynamic Member URL Attribute:** The attribute of the dynamic LDAP group object that specifies the URLs of the members of the dynamic group. With a dynamic group, users are members if they match this URL “rule” (dynamic group members share an attribute or set of attributes). Here is an example URL that specifies a dynamic group that contains any users whose `uid` is in the tree (`o=myldap`) below the `sales` organization: `ldap:///ou=sales,o=myldap??sub?(uid=*)`.
- **User Dynamic Group DN Attribute:** A user attribute indicating its dynamic group membership. If such an attribute does not exist, the provider determines whether a user is a member of a group by evaluating the URLs on the dynamic group. If a group contains other groups, WebLogic Server evaluates the URLs on any of the descendants (subgroups) as well.

LDAP Failover

- The **Host** attribute supports a list of candidate servers for high availability.
- Connection attempts can be made sequentially or in parallel.

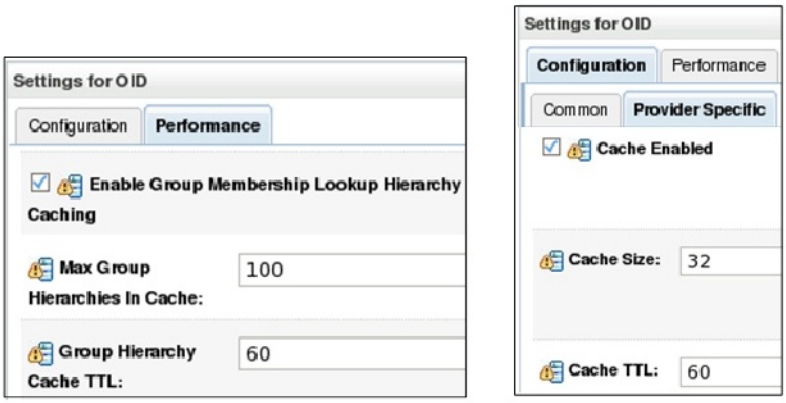


You can configure an LDAP provider to work with multiple LDAP servers and enable failover if one LDAP server is not available. Use the Host attribute to specify the names of the additional LDAP servers. Each host name may include a port number and a trailing comma. Also, configure the following additional attributes:

- **Connect Timeout:** Specifies the maximum number of seconds to wait for the connection to the LDAP server to be established. If set to 0, there is no maximum time limit and WebLogic Server waits until the TCP/IP layer times out to return a connection failure.
- **Connection Retry Limit:** Specifies the number of times to attempt to connect to the LDAP server if the initial connection failed
- **Parallel Connect Delay:** Specifies the number of seconds to delay when making concurrent attempts to connect to multiple servers. An attempt is made to connect to the first server in the list. The next entry in the list is tried only if the attempt to connect to the current host fails. This setting might cause your application to block for an unacceptably long time if a host is down. If the value is greater than 0, another connection setup thread is started after the specified number of delay seconds has passed. If the value is 0, connection attempts are serialized.

LDAP Caching

- All authenticators can cache a group's member list.
- LDAP Authenticators can also cache individual entries.



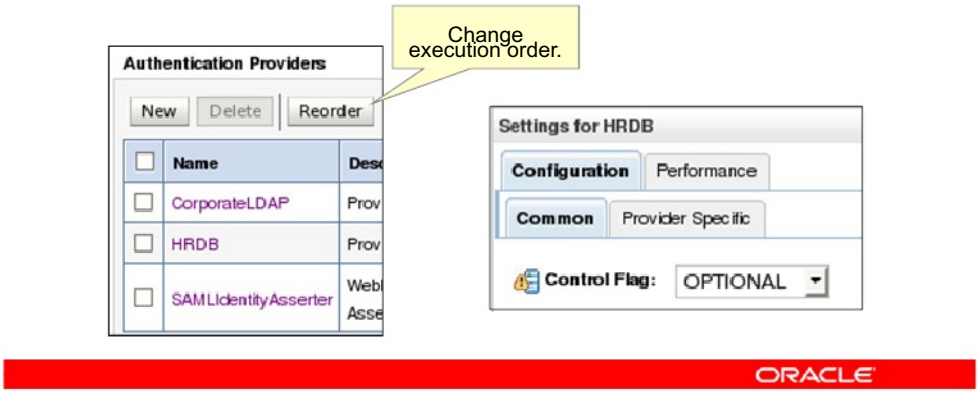
Enabling a cache involves a trade-off of performance and accuracy. Using a cache means that data is retrieved faster, but it runs the risk that the data may not be the latest available.

The time-to-live (TTL) setting specifies how long you are willing to accept potentially stale data. What this value should be depends upon your particular business needs. If you frequently change group memberships for users, then a long TTL could mean that group-related changes will not show up for a while. If group memberships almost never change after a user is added, a longer TTL may be fine. TTL attributes are specified in seconds.

The cache size is related to the amount of memory you have available, as well as the cache TTL. Consider the number of entries that might be loaded in the span of the TTL, and size the cache in relation to that number. A longer TTL will tend to require a larger cache size. For group membership caching, specify the number of groups to cache. For basic entry caching, specify the maximum size of the cache in kilobytes.

Multiple Authentication Providers

- A single security realm can support multiple authentication providers.
- For authenticators, *control flags* determine the processing logic as each provider is executed.



Each security realm must have at least one authentication provider configured. The WebLogic Security Framework supports multiple authentication providers for multipart authentication. Therefore, you can use multiple authentication providers as well as multiple types of authentication providers in a security realm.

The order in which WebLogic Server calls multiple authentication providers can affect the overall outcome of the authentication process. The Authentication Providers table lists the authentication providers in the order in which they will be called. By default, authentication providers are called in the order in which they were configured. Use the **Reorder** button on the **Security Realms** > **Providers** > **Authentication** page in the administration console to change the order in which authentication providers are called by WebLogic Server and listed in the console.

When you configure multiple authentication providers, also use the Control Flag for each provider to control how the authentication providers are used in the login sequence. When additional authentication providers are added to an existing security realm, by default the Control Flag is set to `OPTIONAL`. If necessary, change the setting of the Control Flag and the order of authentication providers so that each authentication provider works properly in the authentication sequence.

Control Flags

Flag	Explanation	Success Action	Failure Action
REQUIRED	Mustsucceed	Executenext provider	Execute next provider, but outcome is: FAIL
REQUISITE	Mustsucceed	Executenext provider	Return control to application with: FAIL
SUFFICIENT	Not required to succeed	Return control to application with: SUCCESS	Execute next provider
OPTIONAL	Not required to succeed	Execute next provider	Execute next provider

- **REQUIRED:** The authenticator is required to succeed. If it succeeds or fails, authentication still continues to proceed down the list. If it has failed, the overall outcome of the authentication is a failure. Use case: This authenticator must succeed, but you still wish to call other authenticators. Perhaps another of the authenticator performs some auditing function.
- **REQUISITE:** The authenticator is required to succeed. If it succeeds, authentication continues down the list. If it fails, control immediately returns to the application (authentication does not proceed down the list). Use case: This authenticator must succeed. If it fails, there is no need to call any other authenticator.
- **SUFFICIENT:** The authenticator is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the list). If it fails, authentication continues down the list. Use case: This authenticator does not have to succeed, but if it does, it is sufficient to validate the user, so no other authenticators need to be called.
- **OPTIONAL:** The authenticator is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the list. Use case: This authenticator does not have to succeed. No matter what it returns, the overall outcome can be success or failure.

The overall authentication succeeds only if all *Required* and *Requisite* authenticators succeed. If a *Sufficient* authenticator is configured and succeeds, then only the *Required* and *Requisite* authenticators prior to that *Sufficient* authenticator need to have succeeded for the overall authentication to succeed. If no *Required* or *Requisite* authenticators are configured for an application, then at least one *Sufficient* or *Optional* authenticator must succeed.

Administration Groups

At least one authentication provider must exist that associates users with groups that have WebLogic Server administrative rights.

Group	Default Capability (via roles and policis)
`Administrators`	Full administrative access to the domain and its applications
`Operators`	View domain configuration, start and stop servers
`Deployers`	View domain configuration, deploy, undeploy, and update applications
`Monitors`	View domain configuration
`AppTesters`	Access applications running in admin mode (servicing administration requests) through the admin port

Often the default authentication provider retains the administrative users, groups, roles and policies; another authentication provider is added for "regular" users, groups, roles and policies.

For easier management of authentication, create groups and add users to them. The default Role Mapping and Authorization providers include policies that grant specific groups different administrative rights in the domain. You can change these policies and roles if desired, but be careful that you do not accidentally make your domain inaccessible. Also, having at least two WebLogic Server administrators at all times helps prevent a single user being locked out, which can make the domain configuration inaccessible until the lockout timeout expires.

The `Administrators` group contains the domain's main administrative user and is assigned to the `Admin` role. Similarly, the `OracleSystemGroup` group contains a user named `OracleSystemUser` and is assigned to the `OracleSystemRole` role.

The remaining administrative groups have no users by default, but are mapped to these roles:

- `Deployers` group: `Deployer` role
- `Operators` group: `Operator` role
- `Monitors` group: `Monitor` role
- `AppTesters` group: `AppTester` role
- `CrossDomainConnectors` group: `CrossDomainConnector` role
- `AdminChannelUsers` group: `AdminChannelUser` role

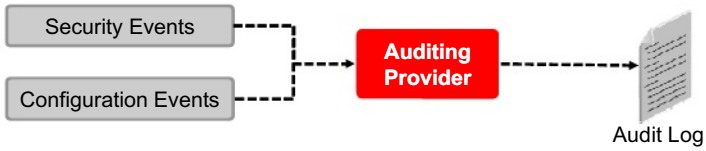
Troubleshooting Authentication

- If you think users are doing things that they should not do, configure an auditing provider.
  - The default auditing provider can be quickly configured.
- Use the server logs.
  - Enable security realm debug flags for more detailed log messages.
- Check the external LDAP authentication provider configuration attributes.
- Use any debug capabilities of the external LDAP Server software.

Auditing Provider

The WebLogic auditing provider:

- Creates a detailed record of all security changes and decisions within a domain in each server's `logs` directory to a file named `DefaultAuditRecorder.log`.
- Can also create a record of all domain configuration changes
- Is not enabled by default



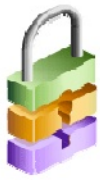
Auditing is the process of collecting and storing information about requested operations and their outcome for the purposes of nonrepudiation. The goal of nonrepudiation is to be able to show that a particular operation is associated with a particular user, so later that user cannot claim someone else performed that operation. In WebLogic Server, an auditing provider performs this function. WebLogic Server includes a default auditing provider, but it is not activated for new security realms. The default auditing provider records information about a number of security requests, which are determined internally by the WebLogic Security Framework. The default auditing provider also records the event data associated with these security requests and the outcome of the requests.

You can also configure the administration server to create audit messages that enable the tracking of configuration changes in a domain. This provides a record of changes made to the configuration of any resource within a domain, as well as invocations of management operations on any resource within a domain. Configuration audit records can be saved to a log file, sent to an auditing provider in the security realm, or both.

All auditing information recorded by the default auditing provider is saved in `<domainname>/servers/<servername>/logs/DefaultAuditRecorder.log` by default. Although you configure the auditing provider at the domain security realm level, each server writes auditing data to its own audit log file in its `logs` directory.

Security Audit Events

- Typical security events:
  - An authentication or identity assertion is attempted.
  - A new role or policy is created.
  - A user account is locked out or is unlocked.
- Security events have the following characteristics:
  - Name
  - Severity (`WARNING`, `ERROR`, `SUCCESS`, and so on)
  - Zero or more "context attributes:"
    - Protocol, port, address
    - HTTP headers
    - EJB method parameters
    - SAML tokens



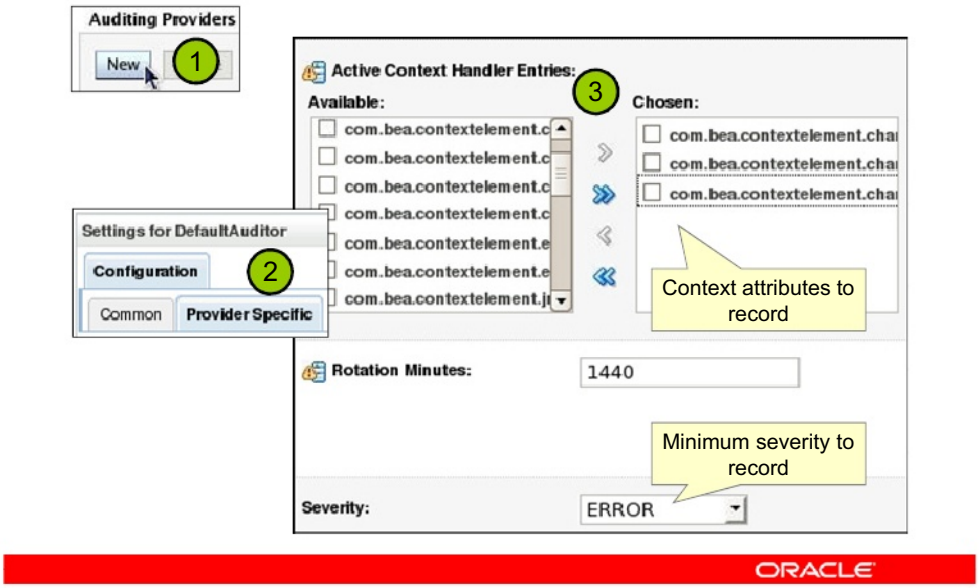
In addition to the events listed in the slide, the default WebLogic auditing provider records the following types of security events:

- When the lock-out on a user account expires.
- A security policy is used and an authorization decision is made.
- A role definition is used.
- A role or policy is removed or "undeployed."

The WebLogic auditing provider audits security events of the specified severity and higher. The severity levels, in order from lowest to highest, are: `INFORMATION`, `WARNING`, `ERROR`, `SUCCESS`, `FAILURE`. You can also set the severity level to `CUSTOM`, and then enable the specific severity levels that you want to audit, such as `ERROR` and `FAILURE` events only.

An audit event includes a context object that can hold a variety of different attributes, depending on the type of event. When you configure an auditing provider, you specify which context attributes are recorded for each event. By default, no context attributes are audited.

Configuring the Auditing Provider



1. After locking the configuration, in the Domain Structure, select **Security Realms** and click the name of the realm you are configuring (for example, `myrealm`). Click the **Providers** > **Auditing** tabs. Click **New**. Give the provider a **Name** and keep the **Type** of `DefaultAuditor`.
2. Select the new provider in the table and click the **Configuration** > **Provider Specific** tabs.
3. Update the following fields, if desired:
   - **Active Context Handler Entries:** Specifies which context attributes are recorded by the auditing provider, if present within an event. Use the arrow buttons to move the Available entries to the Chosen list. The context attributes are things like `HttpServletRequest`, `HttpServletResponse`, `Port`, `Protocol`, `Address`, and so on.
   - **Rotation Minutes:** Specifies how many minutes to wait before creating a new audit file. At the specified time, the audit file is closed and a new one is created.
   - **Severity:** The minimum severity level an event must have to be recorded
   - There are more options if the Severity chosen is `CUSTOM`. There are also options to configure the format of audit log entries under "Advanced."

Security Realm Debug Flags

Flag	Description
DebugSecurityRealm	Trace the initialization of the realm's providers and the loading of initial data from the default store.
DebugSecurityAtn	Trace the authentication and management of users and groups.
DebugSecurityRoleMap	Trace role policy evaluations and results.
DebugSecurityAtz	Trace authorization policy evaluations and access decisions.
DebugSecurityAdjudicator	Trace final authorization decisions.
DebugSecurityUserLockout	Trace the locking and unlocking of user accounts based on the number of invalid login attempts.
DebugSecuritySAML*	Trace the processing and/or generation of SAML tokens.

Multiple SAML security flags

The `DebugSecuritySAML*` flag is listed with an "`*`" to indicate that there are multiple SAML debug flags.

The flags with their scopes:

- `weblogic.security.realm.DebugSecurityRealm`
- `weblogic.security.atn.DebugSecurityAtn`
- `weblogic.security.rolemap.DebugSecurityRoleMap`
- `weblogic.security.atz.DebugSecurityAtz`
- `weblogic.security.adjudicator.DebugSecurityAdjudicator`
- `weblogic.security.userlockout.DebugSecurityUserLockout`

The `DebugSecuritySAML*` flags are found in the following scopes:

- `weblogic.security.saml.atn.*`
- `weblogic.security.saml.credmap.*`
- `weblogic.security.saml.lib.*`
- `weblogic.security.saml.service.*`
- And four more scopes, but replace `saml` with `saml2`

Common LDAP Issues

Typical causes include:

- The wrong base DN, object class, or attribute has been set for users or groups.
- A configured search filter is syntactically valid, but it is semantically incorrect.
  - So, it fails to retrieve the intended users or groups.
- An insufficient "maximum level for nested group memberships" has been set.
  - So, not all group members are found, which means some users are not mapped to their proper roles.
- WebLogic Server does not trust the LDAP server's SSL certificate (and they are set to communicate over SSL).

After a user is authenticated, groups are searched to create a list of groups to which this user belongs. Then role mapping can occur between these groups and roles. If the user does not belong to any groups or if the search criteria are invalid, you will see debug messages:

```
<SecurityDebug> <search(...), base DN & below)>
<SecurityDebug> <Result has more elements: false>
```

The **Max Group Membership Search Level** field, in the configuration of an LDAP authentication provider, specifies how many levels to search when looking for members of a group. (This setting is valid only if **Group Membership Searching** is set to `limited`.) A value of `0` indicates that only direct groups will be found. This means that, when searching for membership in Group A, only direct members of Group A are found. If Group B is a member of Group A, the members of Group B will not be found. A nonzero value indicates how many levels to search. For example, if it is set to `1`, a search for membership in Group A will return direct members of Group A. If Group B is a member of Group A, the members of Group B will also be found. However, if Group C is a member of Group B, the members of Group C will not be found.

If WebLogic Server does not trust the LDAP server's certificate and you have set them up to communicate over SSL, they will not be able to communicate. Perhaps the LDAP server's certificate is from a CA that is not in WebLogic Server's trust keystore.

Quiz

The WebLogic Server default security realm uses this as its security provider store by default:

a. Oracle Database
b. Embedded LDAP Server
c. Derby Database
d. OpenLDAP Server
e. Any Database

Answer: b

Quiz

With LDAP, what does DN stand for?

a. Directory Network
b. Dynamic Name
c. Distinguished Name
d. Directory Name

Answer: c

Quiz

Which of the following is NOT an available authentication provider control flag?

a. SUFFICIENT
b. REQUISITE
c. OPTIONAL
d. ALWAYS
e. REQUIRED

Answer: d

Summary

In this lesson, you should have learned how to:

- Describe the basics of the WebLogic Server security architecture
- Describe basic LDAP concepts
- Configure an external LDAP authentication provider for WebLogic Server

OPTIONAL

Practice 16-1 Overview: Configuring an Authentication Provider

This practice covers the following topics:

- Initializing Apache DS LDAP
- Setting DS LDAP as one of the authentication providers
- Setting the appropriate control flags
- Testing the new authentication provider

17

Backing Up a Domain and Upgrading WebLogic Server

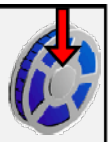
Objectives

After completing this lesson, you should be able to:

- Back up a WebLogic Server domain
- Restore a WebLogic Server domain
- Describe the WebLogic Server upgrade process

Backup and Recovery

Backup
- **Scheduled**
- **At least weekly**
- **Uses different tools for different components**



Recovery
- **Unscheduled (usually)**
- **At least annually (if only to test procedures)**
- **Not necessarily the reverse of backup; it may use other tools**

Backup and recovery:

- Protect against failures of hardware or software, and accidental or malicious changes to the environment
- Guarantee a point of recovery and minimize loss of business availability
- May impact system availability (the system must be offline for an offline backup)
- May include hardware and software

ORACLE

Commonly, the terms “backup” and “recovery” imply the use of secondary media to copy some data for later use. That kind of backup and recovery involves an offline or cold storage of the data such that if an outage occurs, then some process (human or automated) requires some time to get the system back up and running. Alternatively, “redundancy” and “failover” are additional means by which to back up and recover the data in more of an online or warm or hot storage mode, thus reducing, or even eliminating the switchover time. If an outage occurs with redundancy and failover implemented, it is often undetected by the user.

In addition to these features, a media backup plan is essential. The most common problems that require a backup and recovery are to overcome user error or media failure. A more serious problem is when there is a complete loss of the computer hosting the service.

Backup Solution

- Artifacts that need to be backed up include the database, installed products, configured WebLogic domains, WebLogic Server instances, and persistence stores for WebLogic Server TLogs (transaction logs) and JMS resources.
- Use Oracle Recovery Manager (RMAN) to back up database artifacts.
- Use file copy to back up product installations and configured domains, WebLogic Server instances, and persistent stores.
- You can also use the pack utility to back up managed servers.

Oracle Recovery Manager (RMAN) provides backup and recovery for the Oracle database. It provides both a command-line and Enterprise Manager interfaces.

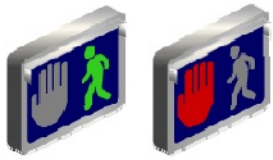
In a Fusion Middleware environment, you should consider backing up the following directories and files:

- Static files (files or directories that do not change frequently)
  - Middleware home (`MW_HOME`): Middleware home contains the FMW products installations including WebLogic Server home, Oracle common home, and Oracle homes for other products. `MW_HOME` can also contain the `user_projects` directory, which contains Oracle WebLogic Server domains and Oracle instance homes, which are not static files.
  - `OraInventory`: In Linux and UNIX, the `oraInst.loc` file points to the directory where the FMW product inventory is located. The inventory is required for patching and upgrading installed components. The `oraInst.loc` file is in the `/etc` directory by default. You can specify a different location for this inventory pointer file by using the `-oraInventories` parameter during the installation of FMW products.

- Dynamic files (the runtime configuration files that change frequently)
  - Domain directories (admin server and managed servers): In most cases, you do not need to back up managed server directories separately because the domain where the administration server resides contains the configuration information for all of the managed servers in the domain, and the pack and unpack utilities can be used to re-create the domains for managed servers.
  - All Oracle instance homes that reside outside of the domain directory
  - Deployed applications, such as `.ear` or `.war` files that reside outside of the domain directory. You do not need to back up application artifacts in managed server directory structures because they can be retrieved from the Administration Server during managed server startup.
  - Database artifacts including any database-based metadata repositories used by Oracle Fusion Middleware. You can use Oracle Recovery Manager (RMAN) to back up an Oracle database.
  - Persistent stores, such as JMS resources and WebLogic Server TLogs (transaction logs)

Types of Backups

- Online:
  - Non-disruptive
  - Possibly inconsistent
    - If backing up takes one hour, the changes made during that hour will not be within the backup and must be tracked
- Offline:
  - Requires all processes to be stopped
  - Relatively easy
- Full:
  - Easier to recover, slower to create
- Incremental:
  - Harder to recover, faster to create



ORACLE

Online

If your environment requires 24x7 availability, you have no choice but to perform online backups. Different components require different tools to perform online backups (also known as hot or inconsistent backups). Inconsistent is not bad in itself; it just means that if the backup takes an hour to complete and you start at 1:00 AM, the files at 1:02 AM will be in a different state than those backed up at 1:59 AM. To accommodate this, there needs to be some kind of online log recording the changes occurring from 1:00 AM until 2:00 AM. This log needs to be incorporated into the recovery, and the logs themselves get backed up at a different time (usually, after they rotate).

Offline

If you can afford to shut down the entire middleware tier (application servers, database, web servers, and so on) for maintenance during some regularly scheduled time, an offline backup is fairly simple (also known as a cold or consistent backup). Using operating system tools such as TAR or ZIP, the backup is guaranteed to be consistent. Make sure you preserve file permissions on UNIX systems.

Full

After the initial installation, or after a major set of patches, a full backup should be performed. Often, this is done before going live, so the system is offline. It is very difficult (if not impossible) to perform a full backup online. If there is a complete loss of a host (for example, a disaster such as a fire or flood), recovery is simple; just copy the backup files to the new host and boot.

Name a backup so as to include the date, for example, `full_backup_2013_04_30.tar`, and keep several generations of them in case you accidentally capture "a problem" in the most recent backup.

Incremental

Considering that the executable files and the configuration files are usually backed up separately, most backups are incremental. Backing up "changes only" may require several sets of backups to perform a full recovery. RMAN can help automate this for databases, especially if the backups are kept online (on disk as opposed to tape).

You can make an incremental backup at the functional level. For example, you can make a WebLogic backup from `<WL_HOME>`, make an instance backup from `<ORACLE_INSTANCE>`, make a database backup from `<ORACLE_HOME>`, and so on. With WebLogic Server, make a backup of all domains and then make backups of individual domains. The disadvantage of doing this is that the backup process will take longer, but the advantage is that the recovery process can be simplified. Alternatively, if you do not make so many different kinds of incremental backups, the backup procedure will complete faster, but now you have complicated and lengthened your potential recovery time. It is a delicate tradeoff balancing storage space versus time versus complexity.

When to Back Up

- Offline backup after the initial domain is created
- Online backups at scheduled intervals
- Online backup after a component changes or the cluster configuration changes
- Online backup before application deployment
- Offline backup before and after patches or upgrades
- Online “database” backups for:
  - LDAP
  - Persistent stores
  - SOA repository

The initial software installation and most patches and upgrades require the servers to be offline anyway, so before and after the patches and upgrades is a good time to perform backups.

Many of the online configuration backups can be automatic by enabling the automatic backup of the domain configuration archive (discussed in the following slides).

The database should be in `archivelog` mode and then backed up with RMAN. In addition, the database should be configured with redundant critical files (for example, control files) and multiplexed critical logs (for example, redo logs). As an added high availability measure, the database can be made completely redundant by using Oracle Real Application Clusters (RAC).

Limitations and Restrictions for Online Backups

- Online backups of WebLogic Server persistent stores are likely to be inconsistent (changes can occur while you are backing up).
  - Database backups can more easily accommodate inconsistencies.
  - File-based stores and OS copies cannot easily accommodate online backup.

None of these restrictions apply to offline backups; they apply only to online backups. In many cases, WebLogic Server has the option to be configured to use either database or file storage for information. Choosing the database is a safer option, but you pay for it with greater complexity and slower performance. If your system uses a database, and the DBA is backing it up, then some additional WebLogic Server tables should not be any additional effort for them. For files such as the configuration XML, application JARs, WARs, or EARs, and properties files, database storage is not an option.

Performing Full Offline Backup

1. Shut down all the processes.
2. Back up <MW_HOME>.
3. Back up the domain.
4. Back up directories from which applications are deployed.
5. Back up the managed server domains on other machines or re-create their domains with the pack/unpack utilities.
6. Back up the instance home for configured system components (like OHS).
7. Back up the database using RMAN.
8. Back up Oracle Inventory.
9. Back up the `oraInst.loc` and `oratab` files (in `/etc`).

To perform a full offline backup:

1. Shut down all processes in Middleware home: 1) system components 2) managed servers 3) admin server 4) Node Managers 5) database 6) database listeners.
2. Back up the Middleware home directory, <MW_HOME>. For example:

   ```
   tar -cpf mw_home_backup_04-12-2013.tar $MW_HOME/*
   ```

3. Back up the domain. For example:

   ```
   tar -zcpf domain_backup_04-12-2013.tarz domain_dir/*
   ```

   Note: It is also possible to use the pack utility.

4. Back up the directory from which applications are deployed. For example:

   ```
   tar -zcpf app_backup_04-12-2013.tarz app_dir/*
   ```

   Note

   - If you deploy applications from a directory inside the domain, this is unnecessary.
   - If you used the pack utility to back up the domain, the deployed applications are already in the JAR file in a directory called _apps_.

5. If the managed servers run on the same computer as the admin server, they use the same domain directories, so do nothing for this step. Managed server domains on other computers can be backed up in the same way as the admin server domain, or they can be recreated with the pack and unpack utilities.
6. Back up the Oracle instance home. The Oracle instance home contains configuration information about system components, such as Oracle HTTP Server or Oracle Internet Directory.
   - For example:

```
tar -cpf instance_home_backup_04-12-2013.tar
                                    $ORACLE_INSTANCE/*
```

7. Back up the database repositories by using the Oracle Recovery Manager (RMAN). If you are doing a full offline backup, you do not need to use RMAN, instead you can just create a TAR file of the database.
8. Back up the OraInventory directory.
   - For example:

```
tar -cpf orainven_home_backup_04-12-2013
                          /u01/app/oracle/oraInventory
```

9. Back up the `oraInst.loc` and `oratab` files, which are usually located in the `/etc` directory.
10. Back up the `oraenv` utility found in the user `bin` directory, for example, `/usr/local/bin`.

Performing Full Online Backup

1. Lock the WebLogic Server configuration.
2. Back up the domain. For example:

```
$> tar -zcpf domain_backup_04-12-2013.tarz /domain_dir/*
```

3. Back up the application directories.

```
$> tar -zcpf app_backup_04-12-2013.tarz /app_dir/*
```

4. If the managed servers are in another location, back up those domain directories.
5. Back up the Oracle instance home.
6. Back up the database with RMAN.

You should perform a backup of runtime artifacts on a regular basis. To back them up:

1. To avoid an inconsistent backup, lock the WebLogic Server configuration, and do not make any configuration changes until the backup is completed.
2. Back up the domain. This backs up Java components such as Oracle SOA Suite and Oracle WebCenter. For example:

   ```
   tar -zcpf domain_backup_04-15-2013.tarz
                                   domain_path/domain_name/*
   ```

   You can also use the pack utility. The advantage of using pack is that it results in a portable JAR file and it also automatically saves all the deployed applications (in a directory in the JAR file called `_apps_`).

3. If the applications are deployed from a directory under the domain directory, you can skip this step. You can also skip this step if you used the pack utility.
4. Back up the managed server directories if they are on a different machine.
5. Back up the Oracle instance home, backing up system components, such as OHS. For example:

```
tar -cpf instance_home_backup_04-15-2013.tar
                                    <ORACLE_INSTANCE>/*
```

   Note: Inform administrators to refrain from configuration changes until after the backup.
6. Back up the database repositories by using the Oracle Recovery Manager (RMAN).

Impact of Administration Server Failure

- Failure of the administration server:
  - Prevents configuration changes in the domain
  - Prevents application deployments
  - Does not affect running managed servers
  - Prevents starting never-started-before managed servers
  - Allows starting previously-started managed servers if Managed Server Independence (MSI) mode is enabled
    - MSI is enabled by default
- Periodically, the managed servers attempt to synchronize configuration data with the administration server.
- When the administration server becomes available, the managed servers get the latest configuration data from the administration server.

When you first start a managed server, it must be able to connect to the administration server to retrieve a copy of the configuration. Subsequently, you can start a managed server even if the administration server is not running.

If a managed server cannot connect to the administration server during its start up, then it uses the locally cached configuration information. A managed server that starts without synchronizing its configuration with the administration server is running in Managed Server Independence (MSI) mode. By default, MSI mode is enabled. However a managed server cannot be started in MSI mode for the first time as the local configuration is not available.

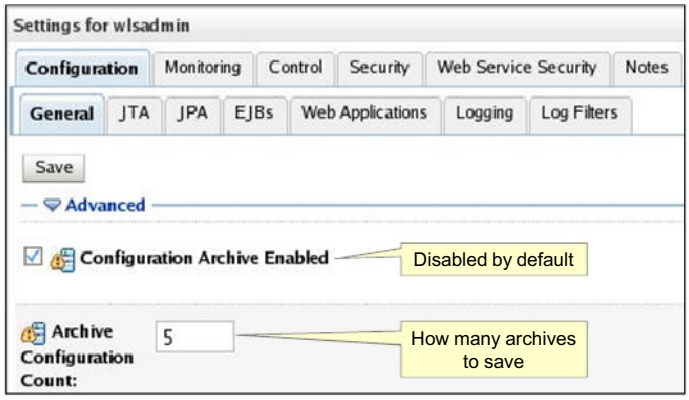
The failure of an administration server does not affect the operation of managed servers in the domain, but it does prevent you from changing the domain's configuration. If an administration server fails because of a hardware or software failure on its host computer, other server instances on the same computer may be similarly affected.

If an administration server becomes unavailable while the managed servers in the domain are running, those managed servers continue to run. Periodically, the managed servers attempt to reconnect to the administration server. When the connection is successful, the configuration state is synchronized with that of the administration server.

For clustered managed server instances, the load balancing and failover capabilities supported by the domain configuration continue to remain available.

Automatically Backing Up a Domain Configuration

Enabling this attribute causes a JAR file of the entire `config` directory to be created each time a configuration change is activated.



Under ***the_domain_name*** > **Configuration** > **General** > **Advanced**, you can enable the automatic backup of the configuration at the domain level. Each startup of the administration server creates two files in the domain directory: `config-booted.jar` and `config-srcinal.jar` . In addition, each activated change of the configuration makes a backup named `configArchive/config-`*n*`.jar`, where *n* is a sequential number. The Archive Configuration Count attribute limits the number of retained configuration JARs. In the example shown, there are never more than five archive files kept. After five backups, older backups are automatically deleted.

You may want to set a higher number such as 10 or 20 for the Archive Configuration Count depending on:

- The available disk space
- The need for backup and restoration
- The time taken for backup and restore activity

Recovery Operations

Some of the common recovery operations include restoring:

- A Middleware home
- An Oracle home
- An Oracle WebLogic Server domain
- The administration server configuration
- A managed server
- An Oracle instance
- Fusion Middleware system component configurations and data

To restore a Middleware home:

- Stop all relevant processes that are related to the domain, such as the administration server, Node Manager, and managed servers.
- Restore the Middleware home directory from a backup.
- Start all relevant processes that rely on the Middleware home.

To restore an Oracle WebLogic Server domain:

- Stop all processes that are related to the domain, such as the administration server and managed servers.
- Restore the domain directory from backup.
- Start all processes that are related to the domain.
  - If you cannot start the administration server, or managed server, you may need to perform recovery of those components.

If the administration server configuration has been lost because of file deletion or file system corruption, the administration server console continues to function if it was already running. The administration server directory is regenerated automatically. However, the security information is not generated. As a result, whenever you start the administration server, it prompts for a username and password. To prevent this, you can recover the configuration.

To recover a managed server :

- If the administration server is not reachable, recover the administration server, as previously described.
- If the managed server fails to start or if the file system is lost, perform the following steps:
  - Recover the Middleware home from the backup, if required, as in the following example:

    ```
    tar -xpf mw_home_backup_04-12-2013.tar
    ```

  - Create a domain template JAR file by using the `pack` utility, as in the following example:

    ```
    pack.sh -domain=path_to_doman/domain_name
    -template=/scratch/temp.jar -template_name=test_install
    -template_author=name -log=/scratch/logs/my.log -managed=true
    ```

  - Copy the JAR file to the managed server computer. Unpack the domain template JAR file by using the unpack utility:

    ```
    unpack.sh -template=/location_of_copy/temp.jar
    -domain=path_to_doman/domain_name
    -log=/scratch/logs/new.log -log_priority=info
    ```

  - Ensure that the application artifacts are accessible from the managed server host. That is, if the application artifacts are not on the same server as the managed server, they must be in a shared location accessible by the managed server.
  - Start the Node Manager on the machine, if it is not already running.
  - Start the managed server using the Oracle WebLogic Server administration console or WLST.

Restoring components:

- You can restore a component's files if they are deleted or corrupted, or if an error occurred during configuration of a component. In these cases, you may want to revert to an earlier version. For Java components, perform the steps to restore a managed server. For System components such as OHS, Oracle Web cache, and so on:
  - Stop the component.
  - Restore the files from the appropriate backup.
  - Start the component.

Note the following key points about recovery:

- Your Fusion Middleware environment must be offline while you are performing recovery.
- Rename the important existing files and directories before you begin restoring files from backup, so that you do not unintentionally overwrite necessary files.
- Although, in some cases, it may appear that only one or two files are lost or corrupted, you should restore the directory structure for the entire element, such as an Oracle instance or a component, rather than just restoring one or two files. In this way, you are more likely to guarantee a successful recovery.
- Recover the database to the most current state, using point-in-time recovery (if the database is configured to be able to do this). This is typically a time right before the database failure occurred.

Directories to Restore

- Binaries (installation directories)
  - Be mindful of preserving group ownership and permissions.
    - These should be read-only for most users.
- Configurations
  - If the last configuration *caused* the problem, recover to a point in time prior to that.
- Log files are:
  - Not required for recovery
  - Created if they do not exist
- Data
  - Database restores data within tablespaces, not directories.
  - RMAN *restore* brings data up to the last backup, then *recover* brings data up to a later point in time.

In most cases, recovery is performed offline. If you think that only one or two files are missing, you may be tempted to recover only those individual files from the backups. However, instead, you should always recover whole directories because there may be other files that are related to these files.

If the directories were backed up from the root, you do not need to worry about the directory you are in when you recover them. The full path information is provided to the operating system, because it is contained in the backup. Restore them as the `root` user, from the root directory, and they will go back to their correct hierarchies. Do not forget the `-p` switch in the `tar` or `jar` command to get the srcinal owner and group information correct.

Recovery After Disaster

- Possible causes of failure:
  - Data loss
  - User error
  - Malicious attack
  - Corruption of data
  - Media failure
  - Application failure
- Recovery depends on the cause:
  - Repair
  - Replace
  - Relocate

If the problem was caused by a minor configuration error, the administrator may be able to reverse the steps and remove the problem without a formal recovery. If the problem requires replacing hardware, restore using full backups. Recovery is complicated when you need to relocate some functions to an existing machine. According to the old configuration (and backups), the functions must be routed to the old name and address of “A,” but now according to the new configuration, the functions need to be routed to the new name and address of “B.”

Recovery of Homes

This applies to recovering a Middleware home, an Oracle home, or an Oracle instance home after data loss or corruption:

1. Stop all processes.
2. Make a new full offline backup as a checkpoint (which can be reused).
3. Change directory to the affected home.
4. Use the OS `copy`, `tar -x`, or `unzip` command for the directories affected.
5. Make a new full offline backup (especially if you have been performing incremental backups up until this point).
6. Restart all processes: A. Database listener B. Database C. Oracle instances (OHS, OID) D. Node Manager E. Administration server F. Managed servers

Ensure that all Fusion Middleware software is stopped so that this is an offline recovery. By performing the two extra backups, you guarantee that you can at least put everything back to the way it was before you tried the recovery.

Recovery of a Managed Server

- If the managed server fails, Node Manager will automatically restart it (if it started it).
- If the files are damaged, you can recover the files in their srcinal places and restart the managed server.
- If the computer is damaged, perform either of the following:
  - Restore the files on a new host with the old computer name by using the OS commands, for example, `copy`, `cp`, `tar`, or `unzip`. (If you backed up by using pack, restore by using unpack.)
  - Restore the files on another host with a different host name by using pack and unpack.

If you used a virtual host name on the old computer, then even if the new computer has a different name, you can still use OS commands to restore the files. Just assign the new computer the same virtual host name.

The srcinal `pack` command that created the remote managed server domain JAR file can be used to recreate the managed server domain in a recovery. The significant configuration and application files are stored at the administration server, so when the managed server is started, it first refreshes all its configuration information and redeploys all its applications from the administration server.

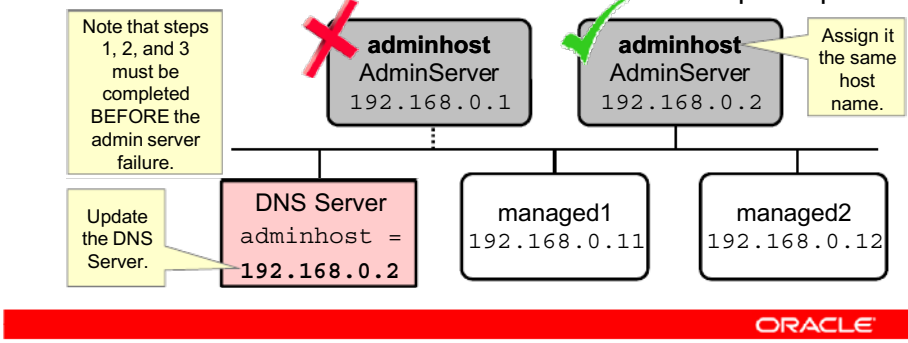
Recovery of the Administration Server

- If the administration server fails, and it was started by using Node Manager (through a WLST script), then Node Manager automatically restarts it.
- If the files are damaged, you can recover the files in their srcinal places and restart the administration server.
- If the computer is damaged, restart the administration server on a new computer.

Restarting the Administration Server on a New Computer

To create a backup of the administration server:

1. Install Oracle WebLogic Server on the backup computer.
2. Copy the application files to the backup computer.
3. Copy the configuration files (or the domain) to the backup computer.
4. Restart the administration server on the backup computer.



If a hardware crash prevents you from restarting the administration server on the same computer, you can recover the management of the running managed servers as follows:

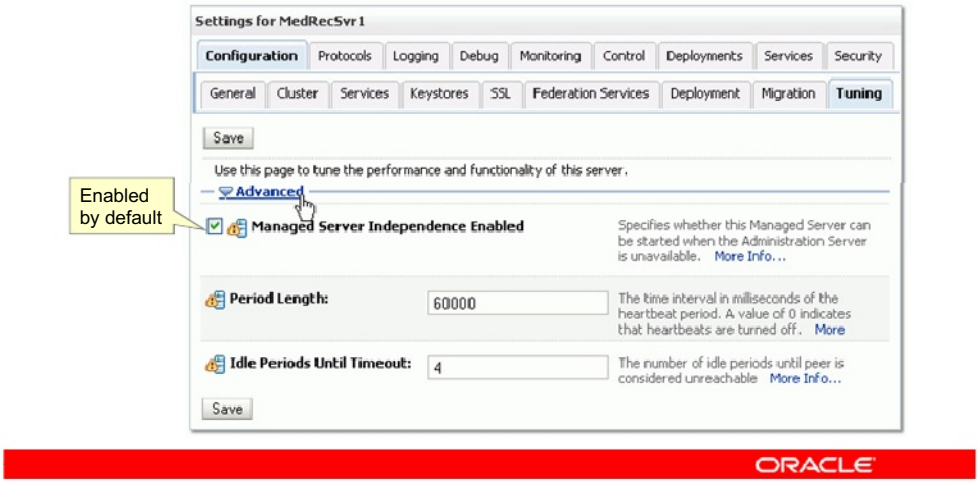
1. Install the Oracle WebLogic Server software on the new computer designated as the replacement administration server.
2. Make the application files available to the new administration server by copying them from backups or by using a shared disk. The files must be available in the same relative location on the new file system as on the file system of the srcinal administration server.
3. Make the configuration and security files available to the new administration computer by copying them from backups or by using a shared disk. These files are located under the directory of the domain being managed by the administration server. An easier option is to copy the entire domain directory to the backup computer.
4. Restart the administration server on the new computer.

Note: In order for managed servers to reconnect after an administration server is restarted on a different IP address, you must have configured a DNS name for the administration server URL that maps to multiple IP addresses. Alternatively, you could have the administration server's listen address set to a virtual host name, and switch the virtual host to the new IP address. Or, if you are using floating IP addresses, assign the administration server's old IP address to the new hardware before restarting the administration server on that hardware.

Important: You cannot have two administration servers running at the same time, both claiming ownership of the same managed servers. Therefore, the administration server standby cannot be a warm standby; it must be a cold standby. The srcinal administration server must be stopped or dead before starting the new administration server on the new hardware.

Managed Server Independence

Managed Server Independence (MSI) mode reduces the urgency to restart a failed admin server.



The administration server is required only for making changes to the active configuration. It is not required for the normal operation of the managed servers, as long as the managed servers have Managed Server Independence mode enabled, which is the default. This allows you time to recover the administration server without any service outages.

As shown in the screenshot, the heartbeat detected between the administration server and the managed servers is, by default, a one-minute time period. After four minutes of not hearing from the administration server, the managed servers become independent. After the administration server is fixed, the heartbeats start up again and the managed servers deactivate their independence, but MSI is still enabled for a future event. These default times can be changed to suit your particular environment.

Upgrading WebLogic Server 11*g* to 12*c*

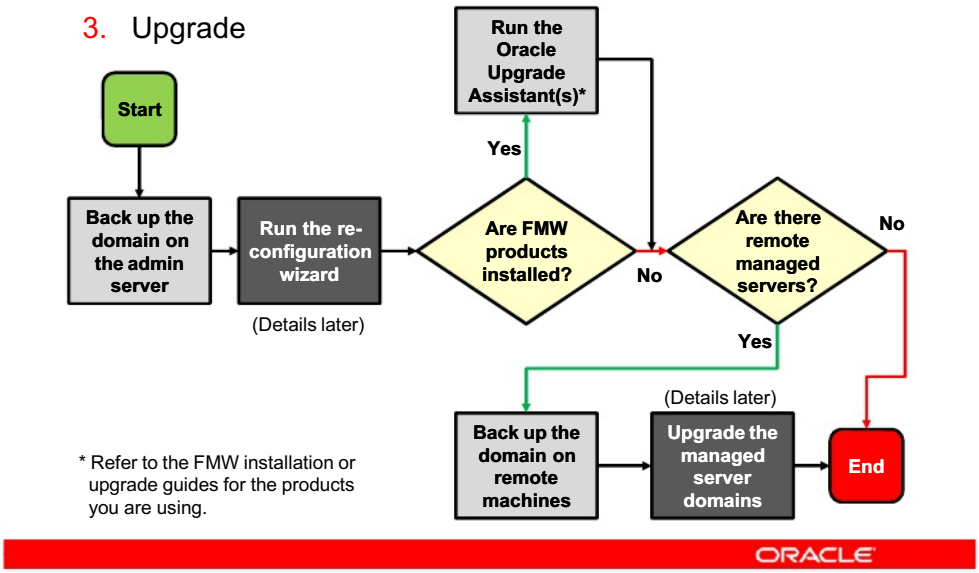
1. Plan the upgrade.
   A. Inventory the environment (admin server, managed servers, applications, external resources, scripts/templates).
   B. Verify that all hardware and software components are supported.
   C. Verify the compatibility of your applications.
   D. Create an upgrade plan.
2. Prepare to upgrade
   A. Undeploy incompatible applications.
   B. Shut down servers.
   C. Back up the environment.
   D. Install new Oracle products.
   E. Prepare remote managed server domains.
   F. Set up environment variables.

ORACLE

1. Plan
   A. The inventory lists all instances of WebLogic Server and the computers on which they reside, the location of all applications, external resources like databases, firewalls, load balancers, and scripts and templates.
   B. Use the *System Requirements and Supported Platforms* spreadsheet to determine if the hardware and software components in the application environment are supported.
   C. Use the *WebLogic Server Compatibility with Previous Releases* appendix to determine any changes that may affect your applications.
   D. Create an upgrade plan. Oracle recommends that you upgrade an application in development environments and use a standard QA, testing, and staging process to move upgraded applications to a production environment. If your environment is complex, you may want to upgrade components in stages.

2. Prepare
   A. In most cases, applications can be run without modification. Use the *WebLogic Server Compatibility with Previous Releases* appendix to see whether your applications use any deprecated or removed features. If so, you may need to modify or undeploy those applications.
   B. Shut down all servers in the environment before upgrading.
   C. Back up the environment:
      i. Back up the domains on the computer where the admin server runs and the computers where the managed servers run. (Note that the Domain Upgrade wizard, which automatically backed up the domain being upgraded, is no longer provided with WebLogic Server.)
      ii. Back up applications and data that reside outside of the domain directories.
      iii. If you do not need a record of log files, you may want to delete them to conserve disk space.
   D. Install the new Oracle products on each computer in the domain.
   E. Prepare the domain directories on managed server computers by copying the following files from the pre-upgraded admin server domain directory to the managed server domain's "root directory:" `/config/config.xml` and `/security/SerializedSystemIni.dat`.
   F. In a terminal window run the `<WL_HOME>/server/bin/setWLSEnv.sh` script.

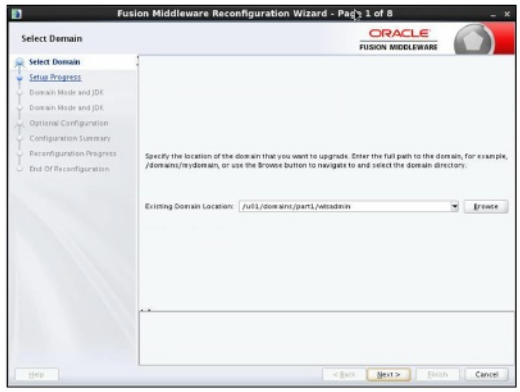
Upgrading WebLogic Server 11*g* to 12*c*



3. Upgrade: The steps in the flow chart that are dark gray are explained further in other slides. Note that if you need to upgrade from a version of WebLogic Server prior to version 10.3.1 (WebLogic Server 10*g*), you must first upgrade to version 10.3.6 (WebLogic Server 11*g*), then upgrade that to version 12.1.x (WebLogic Server 12*c*). You also must use the 11*g* Domain Upgrade wizard to upgrade the domain.

Run the Reconfiguration Wizard

A. In the terminal window, run `<MW_HOME>/oracle_common/common/bin/reconfig.sh`.
B. Go through the wizard screens.
C. Manually finish the Node Manager configuration.



C. To finish the Node Manager configuration:
1. Run `<WL_HOME>/server/bin/startNodeManager.sh`
2. Copy the `<WL_HOME>/common/nodemanager.properties` file from the previous installation into the `<MIDDLEWARE_HOME>/oracle_common/common/nodemanager/` directory of the new installation.
3. Shut down and restart Node Manager.
4. Verify that you can start servers through Node Manager.

Upgrade the Managed Server Domains

A. Ensure that during the preparation phase, you copied these files from the pre-upgraded admin server domain directory to the managed server domain's "root directory:" `/config/config.xml` and `/security/SerializedSystemIni.dat`.
B. Port the reconfigured domain from the admin server computer to the managed server computers with `pack` and `unpack`.

Upgrading WebLogic Server 11*g* to 12*c*

4. Complete post-upgrade procedures:
   A. Re-apply any customizations you had in server start scripts.
   B. Verify and reset file permissions (in Linux, file ownership goes to the user that did the upgrade).
   C. Verify server start options (for example, `JAVA_HOME` and `CLASSPATH` may need to be updated for servers started via Node Manager).
   D. After the environment has been tested, move it to production.

4. Post-upgrade
   A. The Upgrade Wizard does not carry forward any customizations that have been made to the default startup scripts. After the upgrade process is complete, you must customize the default scripts again.
   B. If you backed up the domain directory as part of the upgrade, you should make your backup files secure because they might contain confidential information. During the upgrade process, file permissions are not preserved. If non-default file permissions are set on files, they must be verified and reset. On a UNIX system, ownership and permissions for any new files created during the upgrade process are assigned to the user performing the upgrade.
   C. When you start the administration server, verify the remote server start options, such as `JAVA_HOME`, `MW_HOME`, `BEA_HOME`, and `CLASSPATH`, reference the WebLogic Server 12.1.x installation on the target managed server. This can be accomplished using the administration console on the **Configuration** > **Server Start** screen of the server.
   D. Test your applications in the new environment. If your applications use any deprecated or removed APIs, they can be modified and tested again. Once thoroughly tested, move the environment into production.

Quiz

The administration server of the domain has failed. Can a managed server currently not running be started?

a. Yes, if Managed Server Independence Mode is enabled and the server has been started before.

b. No, a managed server must always contact its admin server when it comes up.

Answer: a

Summary

In this lesson, you should have learned how to:

- Back up a WebLogic Server domain
- Restore a WebLogic Server domain
- Describe the WebLogic Server upgrade process

Practice 17-1 Overview: Backing Up and Restoring a Domain

This practice covers the following topics:

- Backing up a domain
- Restoring a domain