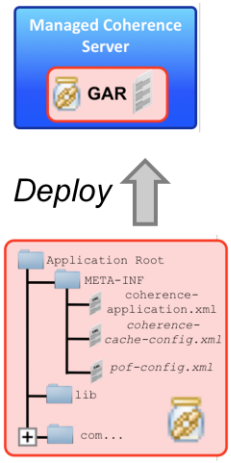
Coherence and WebLogic Server

Oracle WebLogic 12*c* provides comprehensive support for Coherence including:

- Managed Coherence Servers – The ability to configure, start, stop, and otherwise manage Coherence servers natively within a WebLogic environment
- Coherence Grid Archives (GARs) – A JEE-like artifact that encapsulates all the elements of a Coherence application into a single deployable unit



WebLogic 12*c* provides two new features for Coherence.

- Managed Coherence Servers – With earlier versions of WebLogic Server, developers could manage Coherence by using WebLogic Node Manager. However, the process was often difficult and error-prone, and ultimately took advantage of the management capabilities of only a WLS cluster to manage Coherence instances. Coherence was still a stand-alone process. With WebLogic Server 12*c*, Coherence is fully integrated into WebLogic as a first-class member of the cluster, with the ability to take advantage of all aspects of the JEE as well as all aspects of WebLogic Server management.
- Coherence Grid Archives (GARs) – Additionally, Coherence applications are now first-class citizens of a WebLogic Server cluster. Coherence artifacts—including all configuration and deployment information—are packaged into a single file, known as a GAR. A GAR contains everything required to deploy and run Coherence applications. To make this happen, WebLogic now supports an additional container, often referred to as the Coherence container. The Coherence container provides all the same services as an EJB or similar container to Coherence applications. Life cycle, injection, startup, shutdown, class management, isolation, and similar capabilities are provided for Coherence.

The remainder of this lesson details the specifics of these two features, describing the creation, deployment, and management of Grid Archives and managed Coherence servers.

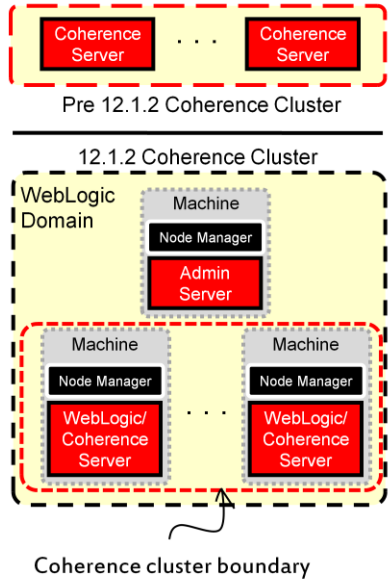
WebLogic Managed Coherence Servers

Operations

The WebLogic Server Container for Oracle Coherence:

- Is a simplified model for managing operational configuration
- Is conceptually similar to other JEE containers
- Is fully integrated and installed with WebLogic and includes Coherence
- Is managed via the configuration wizard, WebLogic console, WLST, JMX, and the Fusion Middleware Control
- Manages Grid Archives (GARs)

More on what a GAR is later



Traditionally, Coherence has been deployed as a JAR, incorporated into a Java application along with a tier of stand-alone cache server JVMs. Embedded Coherence applications, for example from within a WAR, were referred to as clients and the stand-alone cache servers were referred to as servers. The life cycles of the “clients” and cache servers were managed separately, often manually. Application development and deployment in this model was a complicated process involving many moving parts that required custom management processes.

The WebLogic Server Coherence Container for applications provides tight integration between WebLogic Server and Coherence and eliminates the difficulty of managing Coherence instances and applications in a WebLogic environment. This integration allows for simplified and streamlined development and management environments of distributed applications. The Coherence container allows end users to build a new archive type (GAR) that can be deployed and managed via standard WebLogic Server methods. Using the Coherence container:

- Developers can now streamline their build processes to Coherence applications
- Operations departments can now standardize deployment of Coherence instances and Coherence servers.

What Is a Grid Archive (GAR)?

Coherence applications or GARs:

- Are structurally similar to enterprise/web applications but are packaged as a Grid Archive with the extension `.gar`
- Define the elements required for deployment to the WebLogic Coherence container
- Are composed of:
  - A deployment descriptor, `coherence-application.xml`
  - A cache configuration, named in the deployment descriptor
  - Java classes including entry processors, event handlers, filters, application classes, and others
  - Optional support libraries

Defining application name, cache configuration file name, POF configuration, lifecycle handler, and a variety of other information

Coherence applications are packaged into Grid Archive or GAR files. In general, a GAR contains all the artifacts required to service a particular cache or invocation request and mirrors, to a high degree, the structure of other Java EE artifacts such as Web Archives. Specifically, a GAR contains:

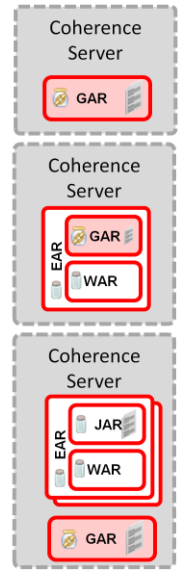
- A deployment descriptor `coherence-application.xml`, which defines the location of a cache configuration as well as a variety of other content such as POF configuration. The deployment descriptor may include references to several other optional elements such as lifecycle handlers.
- A cache configuration, referenced from the deployment descriptor
- Java classes representing the application; these classes typically include entities and other application classes, POF support, entry processors, event handlers, and similar classes
- A set of optional library JAR files
- The entire set of classes encapsulated into a JAR known as a Coherence Grid Archive, which mirrors similar constructs for web applications, enterprise applications, and JAR files in general

GAR files are also deployable directly to stand-alone Coherence, reinforcing their JAR-like structure. Specifically a GAR can be passed to `DefaultCacheServer`. More on this later.

Coherence Application Deployment

Coherence applications can be deployed in one of several ways:

- Stand-alone – GAR is deployed into a server by itself. Analogous to the traditional default cache server.
- Java EE – The application is included as part of:
  - Web Application (WAR) – GAR resources available to WAR only
  - Enterprise Application (EAR) – GAR resources available to WAR and other Java EE resources such as EJBs
- Shared library – GAR resources available to all WAR and EARs located on the server



Coherence applications can be deployed in any of several ways. Deployment models include:

- Stand-alone – A stand-alone deployed Coherence application exposes caches to applications, but does not expose resources directly to WAR or EAR files. Such a deployment is most common when configuring a data tier.
- JAVA EE – GAR files can be deployed as components of a WAR or an EAR file. Such deployments make Coherence resources available to the WAR and/or EAR directly. EJBs, web applications, JSPs, and other JEE artifacts can use dependency injection to access a cache resource. Such a deployment scopes the caches to the EAR itself. An identical deployment, to a different EAR, would result in a similar set of caches, but scoped to the EAR/WAR.
- Shared library – With a shared library deployment, similar to a stand-alone deployment, all WAR and EAR artifacts can access the cache resources directly.

Coherence Container: Benefits

- Simplified installation (Coherence and WebLogic installed together)

Stand-alone Coherence still supported

- Simplified operations management:
  - Configure, manage, and deploy Coherence applications from the WebLogic Server administration console and WLST.
  - Manage Coherence resources centrally.

Within a domain!

WebLogic to Coherence	Coherence benefits to WebLogic

• Instance management
• Dependency injection
• Application lifecycle management
• Scripting support (deployment etc.)
	• Simplified application packaging
	• Multiple deployment models
	  • Stand-alone (GAR)
	  • GAR within web application
	  • GAR within Enterprise application
	  • Shared library

With WebLogic Server 12.1.2, Coherence is provided together with and installed with WebLogic Server. Both WebLogic Server and Coherence provide services to each other.

WebLogic Server provides several services to Coherence, including the following:

- Coherence, separate from WebLogic, cluster, member, and application management and scoping. Applications can be scoped to be stand-alone, WAR, EAR, or shared library scoped. Coherence also provides a cache-level scoping mechanism much like namespaces for caches and data.
- **Packaging** – Available on the system classpath, simplifying packaging
- **Management** – Coherence is now a first-class member of the cluster and fully supported by the console and WLST.
- **Instance support** – Full support for creating and managing Coherence instances
- **Application support** – Full support for application development and runtime life cycle
- **Cluster support** – Full support for defining and managing Coherence clusters

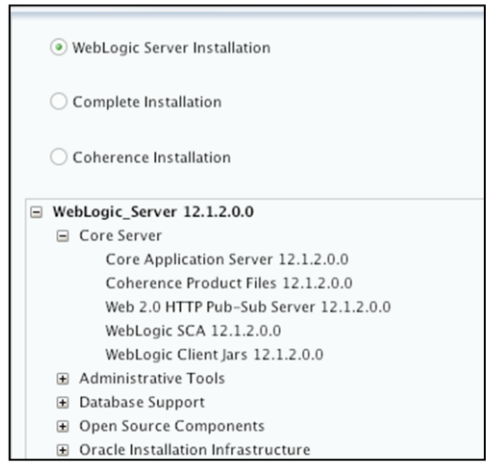
Installing WebLogic with Coherence

WebLogic Server is installed by using:

- Generic Java-based installer
- Platform-specific executable

```
/tmp $ java -jar wls_generic.jar
Extracting files.............
```

Coherence is part of the WebLogic Server core server installation.



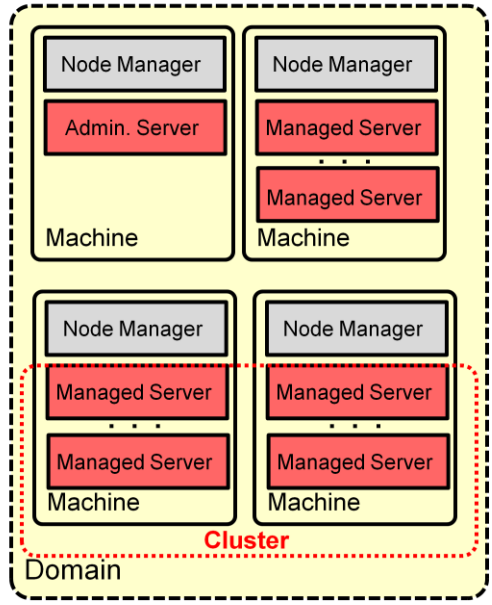
Coherence is installed as part of the normal WebLogic Server installation process. The Coherence product components are installed as part of the standard installation process. Depending on the selected installation type, the following elements are installed:

- WebLogic Server installation – Installs all of WebLogic Server, all of Coherence, as well as the WebLogic Client JAR files
- Coherence installation – Installs all of WebLogic Server and Coherence, but does not install WebLogic Client files
- Complete installation – Installs all of WebLogic Server, Coherence, as well as associated client examples

Terminology: Review

WebLogic includes the following terms and concepts:

- Domain – Group of instances and clusters under unified control
- Administration server – Central configuration controller for the domain
- Managed server – Instance that hosts applications and required resources
- Cluster – Group of managed servers to provide increased scalability and reliability
- Node Manager – Per-machine process used to start, stop, and auto-restart instances



WebLogic defines the following terms, detailed in the following slides.

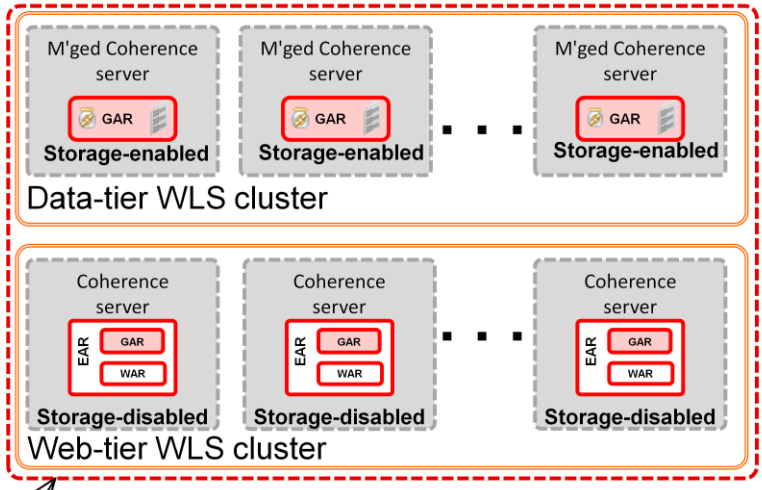
- **Machine** – A physical computer hosting various processes
- **Administration Serve**r – A WLS instance responsible for managing a domain
- **Managed Serve**r – An instance of WLS or Coherence managed by an administration server
- **Node Manager** – A process used by the administration server to manage server instances
- **Cluster** – Any of several types for grouping managed server instances

Coherence Cluster

Coherence clusters consist of multiple managed Coherence server instances.

Managed Coherence server may:

- Be storage-enabled or disabled
- Share configuration and overrides
- Have one or more deployed Coherence applications



Coherence cluster spans multiple WLS clusters.

Coherence clusters are a mechanism for logically grouping sets of managed Coherence servers. In general, Coherence clusters consist of multiple managed Coherence servers, which together form a distributed data grid. Coherence clusters are different from WebLogic Clusters in several ways. First, Coherence clusters often span multiple WebLogic Clusters and are intended to represent Coherence servers, which logically represent a single data grid. Additionally Coherence clusters use different clustering protocols, ports, and are configured separately from WebLogic Clusters. Managed Coherence servers can be associated with a Coherence cluster or with a WebLogic Cluster, which itself is associated with a Coherence cluster.

Typically, a Coherence cluster is associated with one or more WebLogic Clusters, which together represent a tiered architecture, typically having data, web, and proxy tiers. In the sample shown, two WebLogic Clusters—a data-tier cluster and a web-tier cluster—are defined. These two WebLogic Clusters are bound together into a single Coherence cluster. Typically, such a cluster shares a set of Coherence configurations and applications. In the example shown, the same Coherence GAR is deployed to each Coherence managed server, but packaged differently. On the data tier, the application is packaged as a simple GAR and deployed stand-alone. On the web tier, the same GAR is packaged as a component of an EAR and accessible from any other JEE application within that EAR.

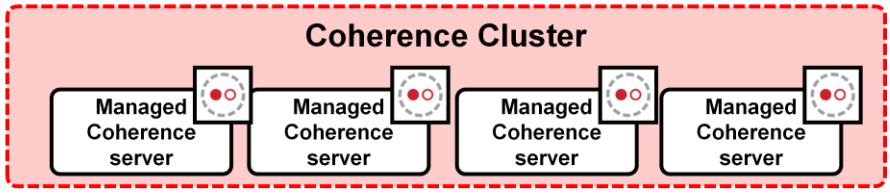
Managed Coherence Server

A domain can have zero or more *managed Coherence servers*

A managed Coherence server:

- Is a WebLogic managed server that encapsulates a Coherence instance
- May be storage-enabled or disabled
- May host Coherence applications
- May host Coherence proxies
- Is normally a member of a Coherence cluster

May be a member of the default Coherence cluster



Managed Coherence servers are server instances that are associated with a Coherence cluster. These servers work together and collectively form a Coherence cluster. Members of a Coherence cluster are historically referred to as Coherence instances, and are referred to as *managed Coherence servers* within a WebLogic installation. However, managed Coherence servers are *not* the same as Coherence servers. Managed Coherence servers are specific to the WebLogic Server Coherence Container integration. The use of Coherence servers, a resource defined in the previous ActiveCache integration, is deprecated. Managed Coherence servers are conceptually no different from earlier Coherence server instances running stand-alone, but there are differences. Some of the differences include:

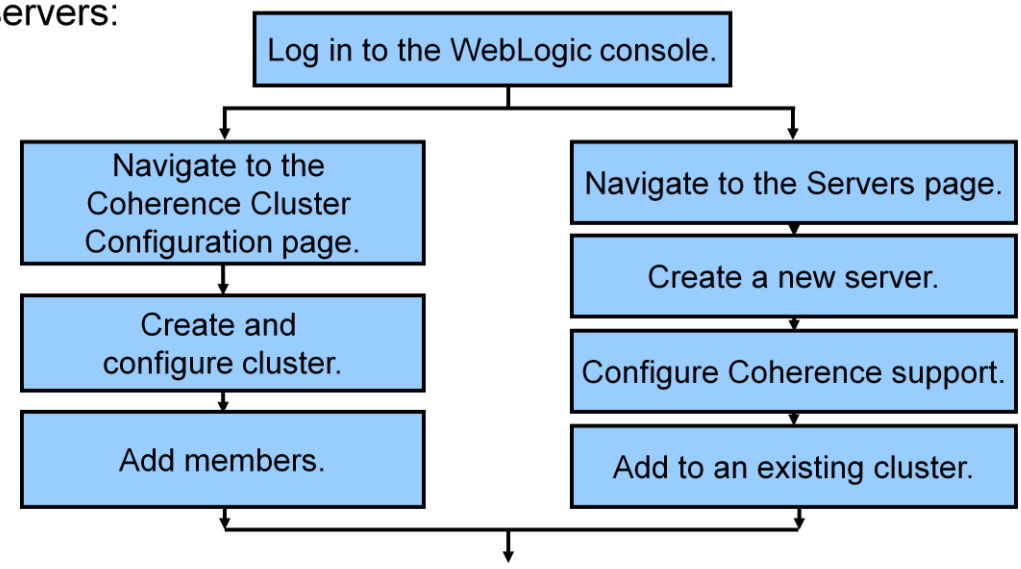
- Managed Coherence servers that are managed within a WebLogic Server domain must not join an external Coherence cluster comprised of stand-alone JVM cluster members.
- Stand-alone JVM cluster members cannot be managed within a WebLogic Server domain.
- The administration server is typically not used as a managed Coherence server in a production environment.

Managed Coherence servers are distinguished by their role in the cluster, and may be:

- **Storage-enabled** – A managed Coherence server that is responsible for storing data in the cluster. Coherence applications are packaged as Grid Archives (GARs) and are deployed on storage-enabled managed Coherence servers.

Working with Coherence Clusters

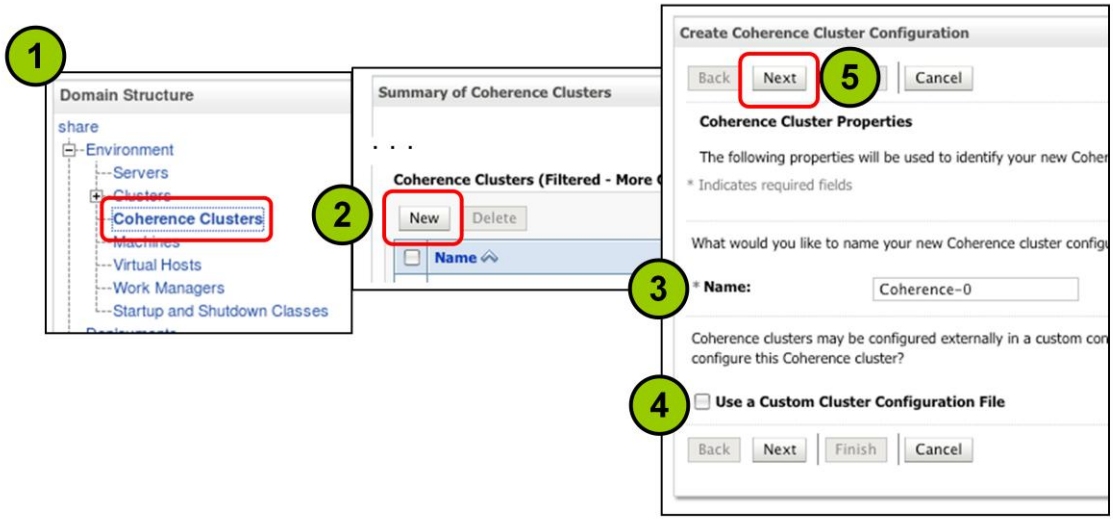
To create Coherence clusters and managed Coherence servers:



Adding Coherence clusters and managed servers is a simple process. Typically, you create a cluster, defining cluster-level characteristics such as required addresses and ports, and then add cluster members. However, it is just as valid to add members and then add them to an existing cluster. Because WebLogic Server instances can also be cluster members, although it is not recommended for performance and scalability purposes, you may simply configure an existing server instance to support Coherence and then add it to a Coherence cluster.

Coherence Cluster Configuration

The WebLogic console supports definition and configuration of Coherence clusters.

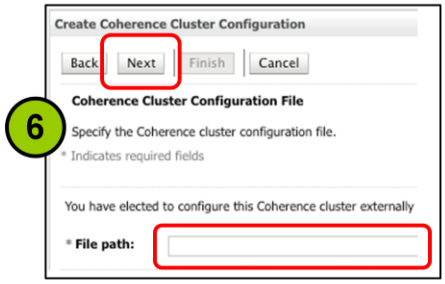


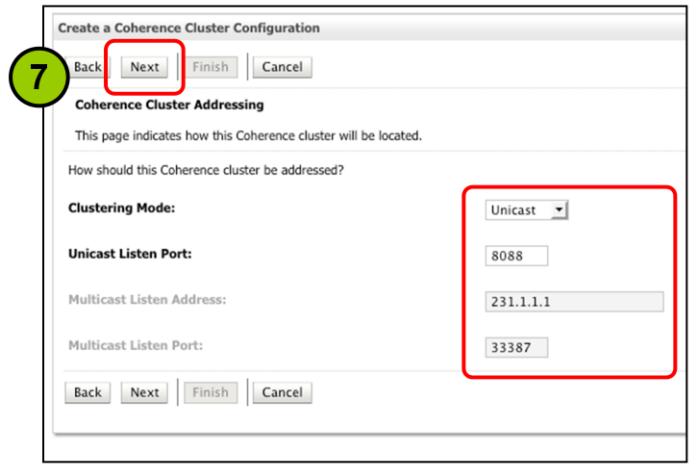
Coherence clusters can be defined within a domain with minimal cluster configuration. Common cluster settings such as multicast address, multicast listen port, logging, time-to-live and others can be configured directly within the console. Additionally, cluster configuration override files can be specified as well.

To define a Coherence cluster:

0. (Not shown in the slide.) Log in and select **Lock & Edit** if required.
1. In the Domain Structure pane, select **Environment > Coherence Clusters**.
2. Click **New** in the Summary of Clusters pane.
3. In the **Create Coherence Cluster Configuration** pane, enter an appropriate name for the cluster.
4. If required, select the **Use A Custom Cluster Configuration File** check box. Note that if this setting is selected, an additional wizard step is added, and you need to enter a fully qualified path to the cluster configuration file, which must be located on the file system of the administration server as well as each managed server.
5. Click **Next**.

Coherence Cluster Configuration





6. Assuming that the Custom Configuration file option was selected, enter a fully qualified path to an operation configuration such as `/opt/coherence/configs/tangosol-coherence-override.xml`. If the setting was not selected on the previous wizard page, then this page is not displayed. Note that it is the responsibility of the deployer to ensure that, if a cache configuration file is specified, the same file exists on all machines and is accessible at the location specified to all members of the cluster.
7. Specify clustering mode, multicast address, and listen port as appropriate and click **Next**.

   Cluster members communicate by using the TCMP protocol, which is different from any other protocols used by WebLogic clustering. TCMP is a IP-based protocol and is used to discover cluster members, communicate, and transfer data. By default, Coherence Clusters use unicast and Coherence Well Known Addresses (WKAs) to discover and communicate with other cluster members. If no WKA members are defined, they are assigned automatically for the cluster. The unicast port of 8088 is used by default. Cluster members may also use multicast; however, multicast is often disabled within a network. See the *Coherence Administration Guide* for more information about multicast, unicast, and WKA configuration.

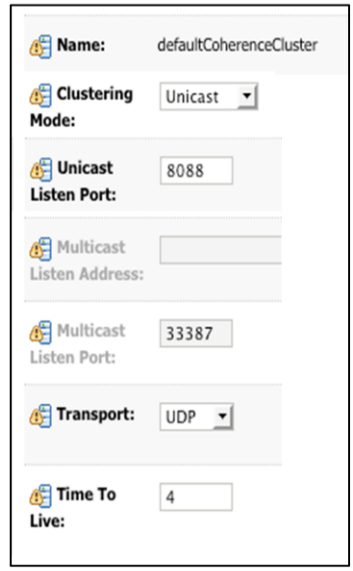
Default Coherence Cluster

WebLogic domains include a default Coherence cluster.

The default Coherence cluster:

- Has a single member, administration server
- Contains sane, traditional defaults
- Can be modified as required
- Is started and runs by default
- Can be deleted or removed safely

The administration server member must be removed before the cluster can be deleted.

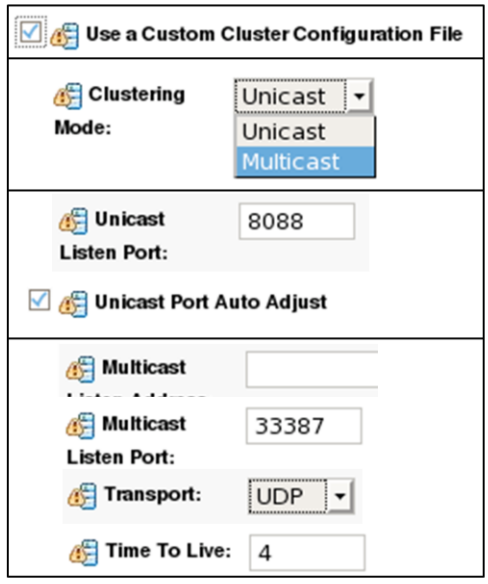


Each Coherence enabled domain includes a default Coherence cluster. The default cluster has a single member, the administration server, and is started and running by default. The default Coherence cluster contains traditional default values that may conflict with other used defined Coherence clusters. The default Coherence cluster may be deleted or modified as needed. However, in order to delete the cluster, first remove any members. After all members are removed, the default Coherence cluster may be deleted.

Coherence Cluster Settings

Coherence cluster settings include:

- **Custom Configuration File**
  - Fully qualified path to a cluster configuration file
- **Clustering Mode**
- **Address Configurations**
  - Unicast listen port and auto-adjust ports
  - Multicast settings including address, port, transport, and time-to-live



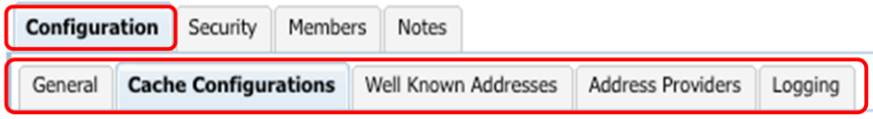
In addition to the Cluster name, which cannot be changed after being created, a Coherence cluster can be modified to specify several default settings. These settings, provided under **Configuration** > **General**, include.

- **Cluster Configuration File:** The Custom Configuration section allows for specifying an overriding cluster configuration file. Coherence itself supports a very large number of configuration options, far beyond those available in the WebLogic console. By enabling this setting and providing a fully qualified path to a configuration file, administrators can concisely control a Coherence cluster. Note that every machine in the cluster must have access to the file using the same path.
- **Clustering Mode:** Unicast or multicast (default: unicast). Supports enabling multicast in environments where multicast is permitted. By default, unicast and WKA are used.
- **Unicast settings:**
  - Unicast Listen Port – The ports on which the server listens for unicast communication. Cluster members use two unicast UDP ports, the default and the next available port.
  - Unicast Port Auto-Adjust – Specifies whether the port automatically increments if the default port is already in use

Additional Coherence Cluster Settings

Coherence clusters support these additional general cluster configuration areas:

- Cache Configurations – Are used to specify a cache configuration by JNDI for injection into GAR modules
- Well Know Addresses – Add, remove, or change managed Coherence servers used for WKA
- Address Providers – Add, remove, or update the addresses and ports used with Coherence WKA specification
- Logging – Enable, disable, and modify settings associated with Coherence logging

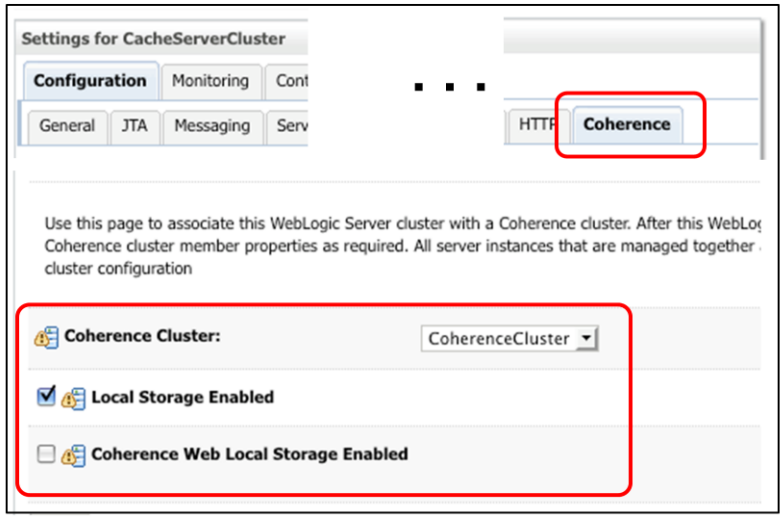


There are several areas where advanced Coherence configuration can be applied. These areas are represented by additional tabs below the general Coherence cluster configuration and include:

- Cache Configurations – A Coherence cache configuration file defines the caches that are used by an application. Typically, a cache configuration file is included in a GAR module. A cache configuration file can be associated with Coherence clusters or managed Coherence servers at run time. In this case, the cache configuration overrides the cache configuration file that is included in a GAR. To override a cache configuration file at run time, the cache configuration file must be bound to a JNDI name. This section allows for the binding of cache configurations to JNDI names.
- Well Know Addresses – The WKA feature allows cluster members to discover and join a cluster by using unicast instead of multicast. WKA members are expected to remain available over the lifetime of the cluster but are not required to be simultaneously active at any point in time. Only one WKA member must be operational for cluster members to discover and join the cluster. This section supports defining WKA members.
- Address Providers – In addition to WKA addresses, address providers may be defined that listen on a given address and port. These providers report the addresses of WKA providers. See the Coherence Cluster Configuration documentation for a complete description.

WebLogic Cluster Coherence Settings

Coherence Managed Servers obtain core settings from the WebLogic Cluster they are a member of:

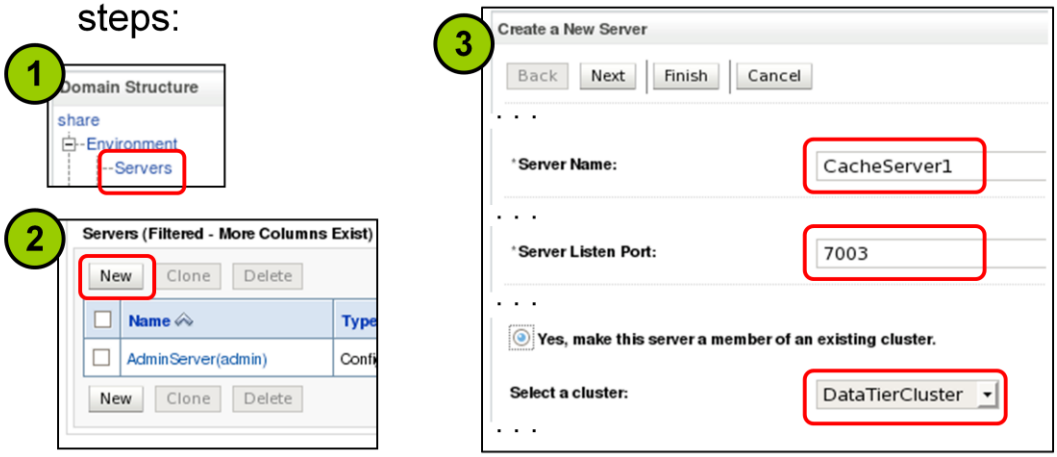


Managed Coherence Server instances obtain some core settings from the WebLogic Cluster they are a member of. This slide shows the Coherence tab for a given WebLogic Cluster. Three settings are available:

- Coherence Cluster – the name of the Coherence Cluster this WebLogic Cluster is a member of.
- Local Storage Enabled – should, by default, members of this WebLogic Cluster be storage enabled. Note that this settings override the storage enable flag associated with a Managed Coherence Server. Defaults to enabled.
- Coherence Web Local Storage Enabled – should, by default, member of this cluster support using Coherence for web session storage? Defaults to disabled.

Creating Managed Coherence Servers

- Managed Coherence servers are created in two steps:
  1. Create a server instance.
  2. Configure the server instance to support Coherence.
- A managed server instance is created in the following steps:



Managed Coherence servers are created in two overall steps:

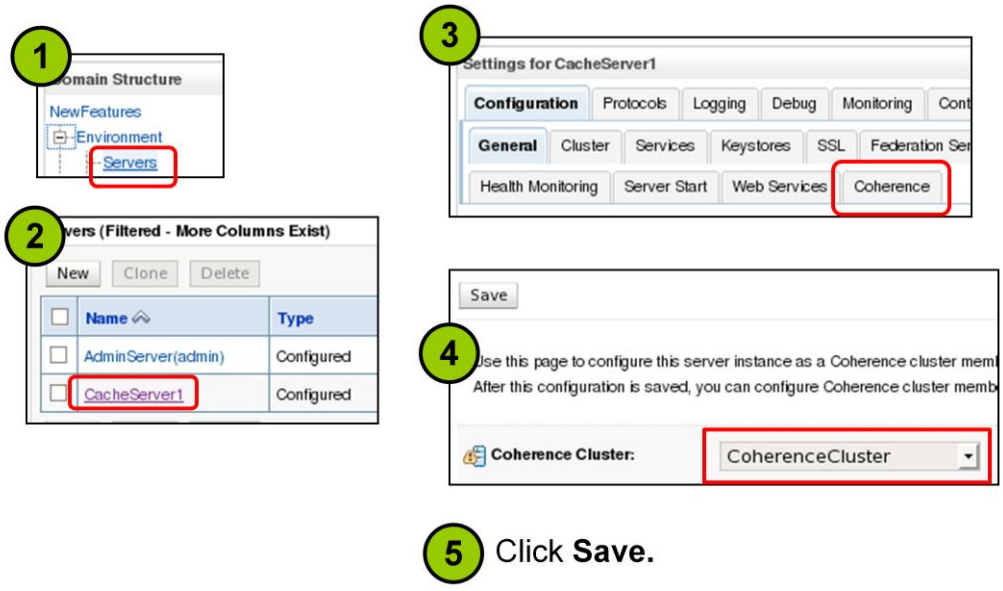
1. First, create an instance of a server, which is normally part of a domain.
2. Configure the newly created server to support Coherence.

To create a server instance:

0. (Not shown) log in to the WebLogic Server console. Note that, depending on how the domain is configured, you may also need to lock the domain for editing and release it when complete.
1. In the **Domain Structure** pane, select the domain to house the server and expand **Environment**; then select **Servers**. The **Summary of Servers** pane is displayed in the right-hand pane.
2. Click the **New** button.
3. In the **Create a New Server** dialog box, enter a **Server Name** and **Server Listen Port**. Optionally, you may select to add the server to an existing cluster, or create a new cluster for this server. In this example, an existing cluster is selected.
4. (Not shown) click **Finish**.
   The newly added server is displayed in the Server list.

Configuring Managed Coherence Servers

To configure a managed Coherence server:



5 Click **Save.**

After a server is created, it must be configured to support Coherence. To initially configure a server instance for Coherence:

1. In the Domain Structure pane, navigate to **domain** > **Environment** and choose **Servers**.
2. From the server list, click a server to configure. The right pane will change to **Settings for Serverxxx**.
3. In the **Settings for Serverxxx** pane, select the **Coherence** tab.
4. Initially, no settings other than the available Coherence clusters are displayed. Using the drop-down list, select a previously configured Coherence cluster. Note that there is a defaultCoherenceCluster available for use as well. This Coherence cluster is created when the domain is initially created. However, the default Coherence cluster can be removed or have its name changed during domain creation and, as such, may not exist.
5. Click **Save**.

Additional Coherence Server Settings

Coherence server settings include:

- **Unicast Settings**
  - Unicast Listen Address
  - Unicast Listen Port
  - Unicast Port Auto Adjust
- **Storage Settings**
  - Local Storage Enabled
  - Coherence Web Local Storage Enabled
- **Member Identity Settings**
  - Site Name
  - Rack Name
  - Role Name

Local storage flag only applies if NOT a member of a WLS cluster

After a Coherence cluster name has been selected and saved for the managed Coherence server, several additional settings become available. These settings fall into three categories:

Unicast Settings:

- Unicast Listen Address – The address on which the server listens for unicast communication. An address must be specified if the computer has multiple IPs or NICs. The field also supports Classless Inter-Domain Routing (CIDR) notation, which uses a subnet and mask pattern for a local address to bind to rather than a specific IP address. A CIDR example might include: 192.168.12.0/23, which applies the network mask 255.255.254.0 to the 192.168 network, starting at 192.168.12.0. This notation represents the address range 192.168.12.0 – 192.168.13.255.
- Unicast Listen Port – The ports on which the server listens for unicast communication. A cluster member uses two unicast UDP ports, the default port and the next available port.
- Unicast Port Auto Adjust – Specifies whether the port automatically increments if the default port is already in use

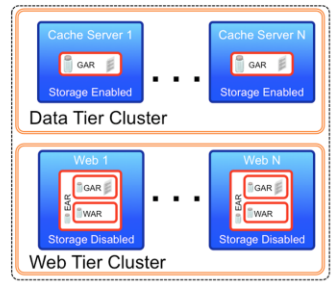
Storage Settings:

- Local Storage Enabled – Specifies whether a managed Coherence server stores data
- Coherence Web Local Storage Enabled – Specifies whether a managed Coherence server stores HTTP session data

Coherence Deployment Tiers and Topologies

Coherence applications are deployed to WebLogic clusters, referred to as a tier.

Combinations of tiers are referred to as topologies.



Tiers typically include:

- Data tier – A WLS cluster of storage-enabled servers, hosting a Coherence application and serving cache data requests
- Web tier – A WLS cluster of storage-disabled servers, accessing the data tier and hosting WAR/EARs
- Proxy tier – A WLS cluster of storage-disabled servers hosting Coherence proxy servers
- Client tier – Stand-alone applications (Java/C++ …) and REST

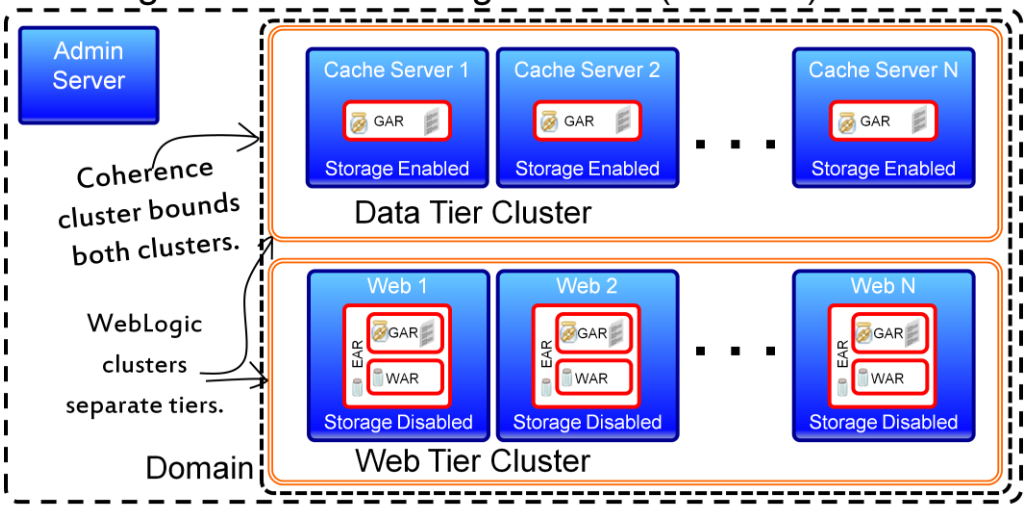
Deployment topologies are mechanisms for defining and accessing data from within WebLogic applications. Each has its advantages and disadvantages and can be used in a variety of situations. Deployment topologies typically include three specific tiers, each dedicated to a specific function:

- Data tier – A set of WebLogic Server instances dedicated to running Coherence applications packaged as a stand-alone GAR. Typically, data tier members, referred to as cache servers, are members of a dedicated WebLogic cluster. However, cluster membership is not required. Depending on the GAR being loaded, members constitute one or more Coherence clusters. Members of the data tier are always running storage-enabled.
- Web tier – A set of WebLogic Server instances dedicated to running Enterprise or web applications, which themselves contain a Coherence application GAR. Typically, web tier members are considered cache clients, and are members of specific Coherence clusters based on the GAR files loaded. Cache clients are typically members of a dedicated WebLogic cluster, but need not be. Members of the web tier almost always run storage-disabled. Note that web tier members may or may not run proxy server instances.

Best Practice:
Single Data with Web Cluster Topology

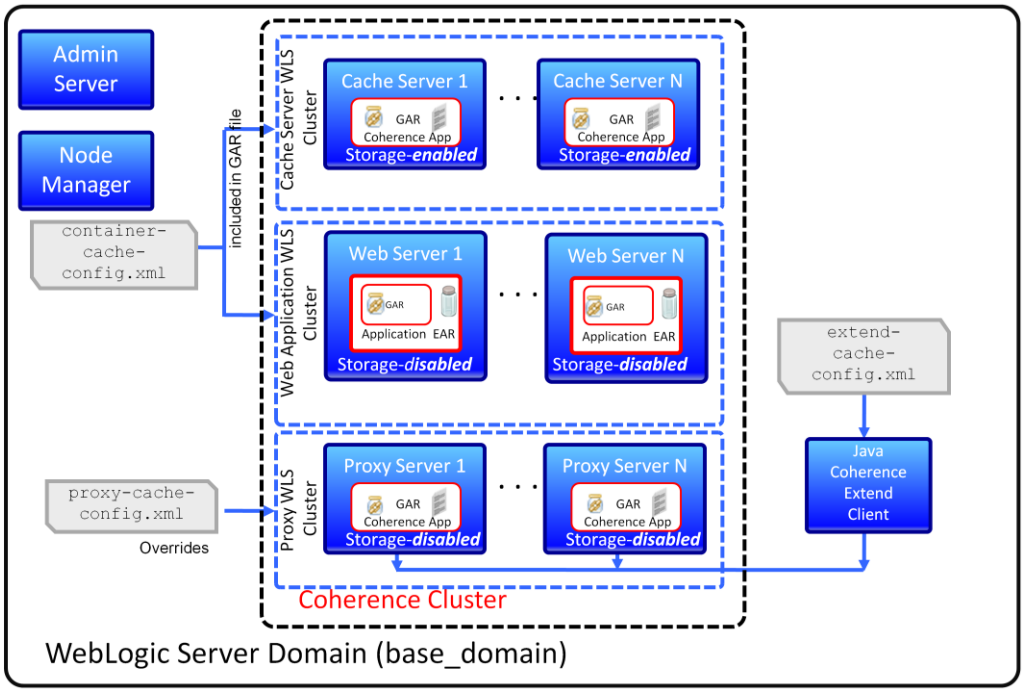
With the data and web cluster topology:

- Storage-enabled Coherence cluster (data tier)
- Storage-disabled WebLogic cluster (web tier)



The data and web cluster topology separates the data and web applications into two WebLogic clusters, each with a different purpose. Data is housed in the data tier cluster, which is composed of three or more data servers, each storage-enabled and running the Coherence application. The web tier cluster then houses the web application and runs storage-disabled. As with the previous example, the web tier and data tier use identical cache configurations read from the deployed GAR file.

Coherence Deployment Topology



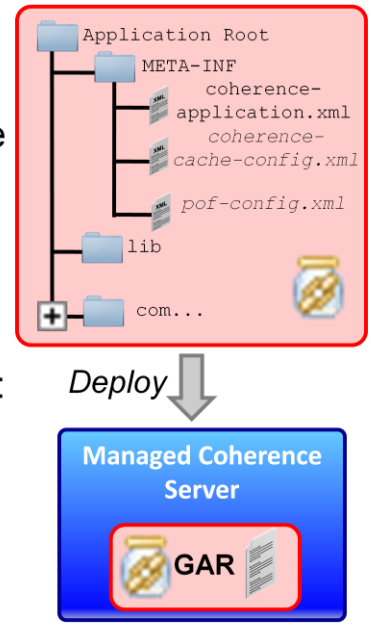
This example shown is a common topology based on three tiers. The tiers are selected as follows:

- Cache Server Tier – running storage enabled with only the GAR application itself. No applications other then those related to Coherence should be running on this tier.
- Web Server Tier – running storage disabled and loading an enterprise application which contains the identical GAR to that deployed in the Cache tier. These instances should run storage disabled to avoid rebalancing as they start and stop, but also to avoid mis-matching memory requirements due to a mixed Web/Coherence application mix.
- Proxy Tier – Also running storage disabled, loading the same application as the WebApplication tier but overriding the cache configuration to specify proxy specfic configuration.

Coherence Application Grid Archives

Coherence Grid Archives (GARs):

- Follows the Java EE model
- Packages into a single distributable file
  - Application, cache, and POF configuration
  - Entity and support classes
  - Other optional libraries
- Leverages WebLogic infrastructure for:
  - Deployment – Console and WLST
  - Management dependency injection
  - Lifecycle events



Coherence applications are packaged into Grid Archive or GAR files. In general, a GAR contains all the artifacts required to service a particular cache or invocation request and mirrors, to a high degree, the structure of other Java EE artifacts such as Web Archives. Specifically, a GAR contains:

- A deployment descriptor that includes the location of a cache configuration as well as a variety of other configurations
- A cache configuration, referenced from the deployment descriptor
- Java classes representing the application; these classes typically include the entities and other application classes, POF support, entry processors, event handlers, and similar classes
- A set of optional library JAR files

The entire set of classes is encapsulated into a JAR knows as a Coherence Grid Archive, which mirrors similar constructs for web applications, enterprise applications, and JAR files in general.

GAR Structure

GAR files are structured very similarly to EAR/WAR files.

Directories and Files	Description
Application Root	Application Root Directory
META-INF coherence-application.xml *coherence-cache-config.xml* *pof-config.xml*	Traditional manifest directory. Contains `coherence-application.xml`, cache configuration, POF configuration, and other files
lib	Optional support libraries
com...	Application classes

Coherence application GAR files are created by using the JAR tool and have a format and structure similar to a web application. To be recognizable by WebLogic Server, the application must have extension `.gar` and must include a valid deployment descriptor and be packed with JAR or JAR-like tools.

Coherence applications are structured as:

- `META-INF` – This folder contains the required coherence-application.xml file as well as potentially other files. Note that for convenience, it is common to include a cache configuration, POF configuration, and other files. However, it is not a requirement to group configuration files into this directory. For backward capability, developers might consider placing the cache configuration in the root folder.
- `META-INF\coherence-application.xml` – The application deployment descriptor that must exist in the META-INF directory. This file specifies the location of the cache configuration as well as potentially other files.
- `META-INF\coherence-cache-config.xml` – Coherence cache configuration
- `META-INF\pof-config.xml` – Optional Coherence POF configuration
- `lib` – An optional directory containing support JAR files
- `Classes` – Shown as “com…”, a folder containing application classes such as entity classes, entry processors, and all other components of the application itself

Deployment Descriptor

coherence-application.xml

The Coherence application deployment descriptor defines:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<coherence-application>
  <cache-configuration-ref>
       META-INF/cache-config.xml
  </cache-configuration-ref>

  <pof-configuration-ref>
       META-INF/pof-config.xml
  </pof-configuration-ref>
  <application-lifecycle-listener>
      <class-name>. . . </class-name>
  </application-lifecycle-listener>
  <configurable-cache-factory-config>
     <class-name>. . . </class-name>
  <configurable-cache-factory-config>
</coherence-application>
```

Required: A relative reference to the cache configuration used by this application

Optional elements:

- POF configuration
- Lifecycle handler
- Configurable cache factory

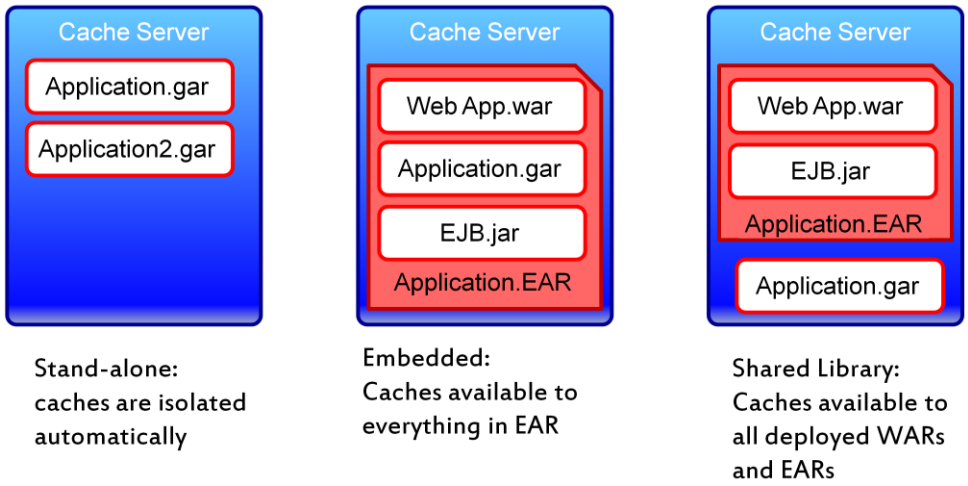
Every Coherence application must include a deployment descriptor that provides information required by the Coherence container to load and process the application. The deployment descriptor must be named `coherence-application.xml` and must be located in the `META-INF` directory of the GAR.

Deployment descriptors include any of four child elements of the `coherence-application` element. The elements are, in order:

- `cache-configuration-ref` – (Required) contains the relative path to the applications cache configuration definition. Note that this element supports the `override-property` attribute, which defines the name of an override property that can be used to specify an alternative configuration file.
- `pof-configuration-ref` – (Optional) contains the relative path to the application's POF configuration definition
- `application-lifecycle-listener` – (Optional) contains a `class-name` element, which itself contains the fully qualified name of a class that implements the `com.tangosol.application.LifecycleListener` interface. This is covered in a later section.
- `configurable-cache-factory-config` – (Optional) contains a `class-name` element, which itself contains the fully qualified name of a class that implements the `com.tangosol.net.CacheFactory` interface

GAR Deployment Modes

GAR modules can be deployed to a managed Coherence server in several ways, including the following:



Coherence applications packaged into GAR files can be deployed in any number of ways. The most common include:

- Stand-alone GAR – An application can be packaged and deployed as a stand-alone GAR. Such applications, and the caches they imply, are accessible to themselves only. Two different applications cannot access each other's cache definitions. Applications including the same cache names are scoped by the application name, or by using the cache configuration `<scope>` element. A single instance of each cache exists per application.
- Embedded within an EAR – Applications are packaged and deployed as part of a stand-alone EAR. Web applications, EJBs, and similar artifacts can, via injection, access caches. Multiple EARs containing the same GAR are isolated from each other both from a class-loading and cache perspective. A single instance of the cache exists for each EAR.
- As a shared library – The GAR file is deployed as a shared library. All EARs and WARs can access the application. A single instance of the cache exists per deployed server.

Note that Coherence can also be deployed as part of a WAR. See the Coherence development documentation for directions about packaging GARs within a WAR.

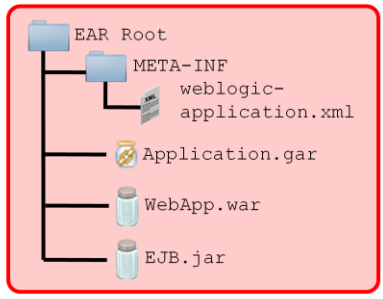
Packaging GARs Inside EARs

A GAR must be packed into an EAR module to be used by other modules in that EAR.

To include a GAR in an EAR:

1. Assemble the EAR with the GAR at the root of the EAR along with any other modules.
2. Add a module entry for the GAR to `weblogic-application.xml`.

```
<?xml version = '1.0'>
<weblogic-application . . .>
    <module>
      <name>Application</name>
      <type>GAR</type>
      <path>Application.gar</path>
   </module>
</weblogic-application>
```



Coherence applications, packaged as GAR files, are packaged inside an EAR by performing the following:

1. Ensure that the GAR is packaged at the root of the EAR as shown in the slide.
2. Modify `weblogic-application.xml` to specify a module element containing the `name`, `type`, and `path` to the GAR.
3. Package and deploy the EAR normally.

Application Isolation and EARs

With Enterprise applications, the GAR module name must match the stand-alone deployment name.

```
<?xml version = '1.0'>
<weblogic-application . . .>
    <module>
      <name>ExampleGAR</name>
      <type>GAR</type>
      <path>ExampleGAR.gar</path>
   </module>
</weblogic-application>
```

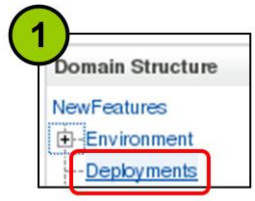
Data is shared between applications when module name matches an external GAR.

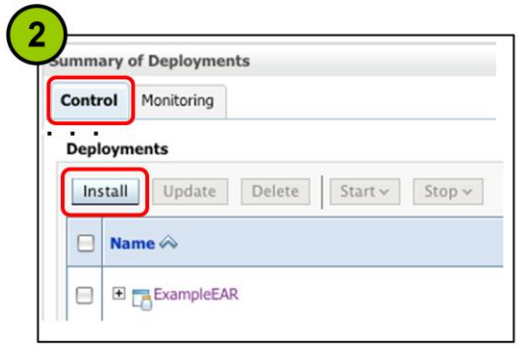
In order for data to be shared correctly between an EAR with an embedded GAR and a stand-alone GAR, the EAR must be correctly configured. Two criteria must be met:

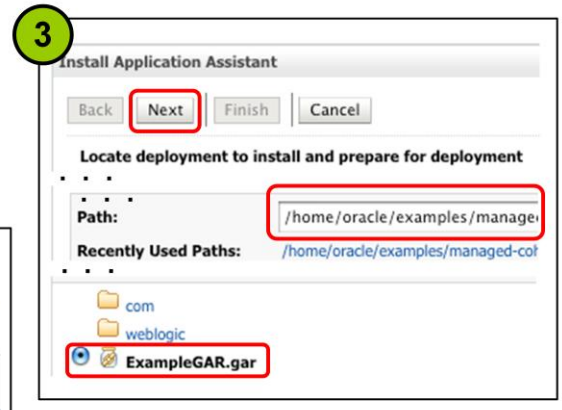
- First, the EAR must contain the same GAR as the one deployed stand-alone.
- Additionally, `weblogic-application.xml` must specify a matching name for the GAR. In the example shown, `ExampleGAR.gar` was deployed stand-alone and within an EAR. The EAR then correctly referenced the GAR module and named it exactly as the stand-alone GAR was named.

Deploying Coherence Applications

To deploy a Coherence application GAR file:



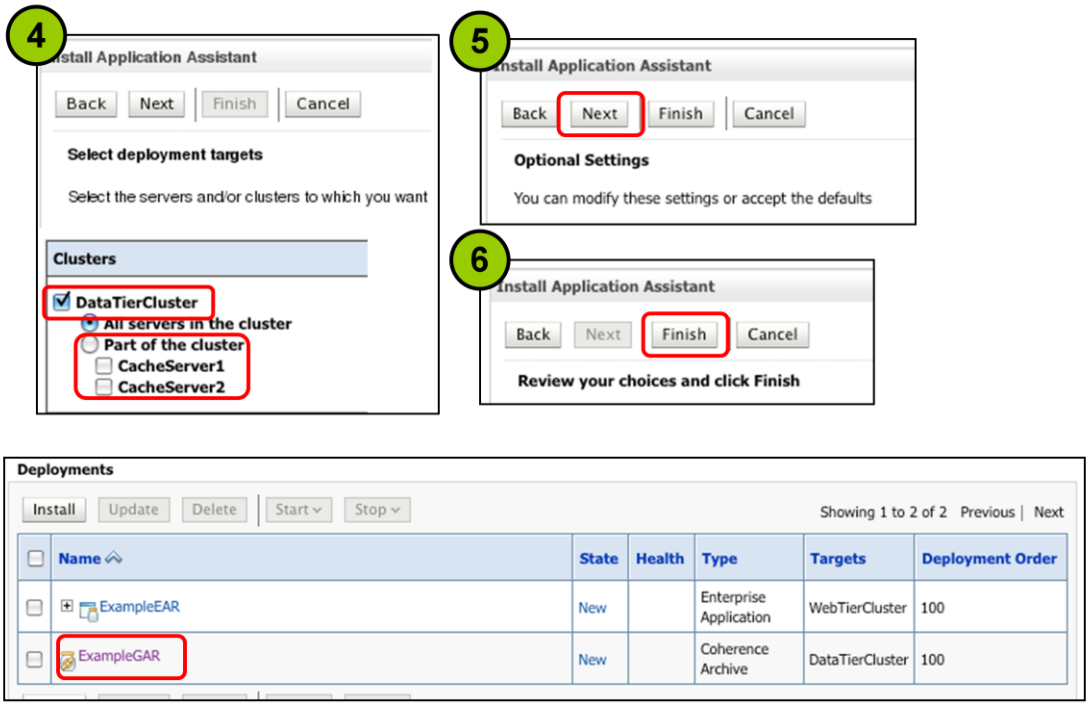




Coherence applications are deployed similarly to other JEE artifacts such as EARs or WARs. To deploy a Coherence application:

1. In the Domain Structure pane, navigate to the domain and then **Environment** > **Deployments**.
2. In the **Summary of Deployments** pane, click **Install**. Note that the **Control** tab is displayed by default. The pane will be replaced by the deployment wizard.
3. Enter the fully qualified path to the directory holding the GAR. Note that you may enter a partial path and then browse the directory structure to locate a directory.
   Select the Coherence application (GAR) and click **Next**.

Deploying Coherence Applications



4. Select a target for the deployment, which may be an entire Coherence cluster or a portion of a cluster, and click Next.
5. Modify the Optional Settings and click **Next**.
6. Review the completed deployment, use the Back button to return to a prior portion of the wizard and correct any errors, and then click Finish to complete the deployment.

The newly deployed Coherence application will then be displayed in the **Summary of Deployments** pane in the **Deployments** section.

WebLogic Scripting Tool (WLST)

- The WLS command-line tools are useful:
  - For automating common administration activities
  - As an alternative to the administration console
  - When graphical tools are not supported
- WLST provides a command-line interface for:
  - Creating new WLS domains
  - Retrieving and updating WLS domain configurations
  - Deploying applications
  - Obtaining runtime server statistics

WLST can be used for creating Coherence clusters, as well as deploying GARs and EARs containing GARs!

The WLST is a command-line scripting environment that you can use to create, manage, and monitor Oracle WebLogic Server domains. It is based on the Java scripting interpreter, Jython. In addition to supporting standard Jython features such as local variables, conditional variables, and flow control statements, WLST provides a set of scripting functions (commands) that are specific to Oracle WebLogic Server. You can extend the WebLogic scripting language to suit your needs by following the Jython language syntax.

Running WLST Scripts

- Use the `setWLSEnv.sh` or `setDomainEnv.sh` scripts to initialize the `WLST` environment.
  - If no script file is supplied, `WLST` runs in interactive mode.
- Use the `execfile()` command to run additional scripts.

```
$ . .../setDomainEnv.sh
$ java weblogic.WLST [scriptfile.py]
```

To support SSL connection to a server:

```
$ java -Dweblogic.security.SSL.ignoreHostnameVerification=true
-Dweblogic.security.TrustKeyStore=DemoTrust weblogic.WLST
```

Command Line

Add the Oracle WebLogic Server classes to the `CLASSPATH` environment variable and `WEBLOGIC_HOME/server/bin` to the `PATH` environment variable. You can use the `WEBLOGIC_HOME/server/bin/setWLSEnv` script to set both variables. (This script is also used indirectly by other domain scripts, such as `setDomainEnv`.)

For convenience, the following additional options are available for launching WLST in interactive mode:

- The `WEBLOGIC_HOME/common/bin/wlst` script
- On Windows, the Tools > WebLogic Scripting Tool from the Start menu

Use the following system properties if you plan to connect WLST to an Oracle WebLogic Server instance through an SSL listen port, and if the server instance is using the WebLogic demonstration SSL keys and certificates:

```
-Dweblogic.security.SSL.ignoreHostnameVerification=true
-Dweblogic.security.TrustKeyStore=keystorename
```

Use this option to run a WLST script, where `filePath.py` is an absolute or relative path name for the script: `[-i] filePath.py`

By default, WLST exits (stops the Java process) after it executes the script. Include `-i` to prevent WLST from exiting. Instead of using this command-line option, you can use the following command after you start WLST: `execfile('filePath.py')`

WLST: Creating Coherence Clusters

To create a new Coherence cluster by using WLST:

```
connect()
edit()
startEdit()

cohSR = create('clustername',
                    'CoherenceClusterSystemResource')

cohBean  = cohSR.getCoherenceClusterResource()

cohCParams = cohBean.getCoherenceClusterParams()
. . .

activate()
disconnect()
```

Connect to a domain, and start editing.

Create a new Coherence cluster resource.

From the resource obtain its MBean.

Use the resource MBean to get the cluster's parameter MBean.

Use this MBean to set parameters for the cluster.

Complete the edit and disconnect from the domain.

To create a new Coherence cluster confirmation by using WLST:

1. Connect to the machine normally. In this example the `connect()`, `edit()` and `startEdit()` commands are used to inform the administration server that an edit of the domain configuration is taking place.
2. Using the `create()` command, create an object of type `CoherenceClusterSystemResource` with appropriate name. In the example shown, the name of the cluster is `clustername`. Note that the `create` command returns a system resource.
3. Using the system resource returned in the previous step, obtain the Coherence cluster Mbean by using the `getCoherenceClusterResource()` command. The MBean representing a Coherence cluster can be used to return its parameters MBean. Using `getCoherenceClusterParams()` return the Coherence cluster params Mbean, which can be used to modify the clusters parameters.
4. When all modifications have been made, use the `activate()` and `disconnect()` commands to finalize the edit and disconnect from the server.

Modifying Coherence Cluster Parameters

WLST can be used to modify a new or existing Coherence cluster. To modify an new Coherence cluster configuration:

```
. . .
cohCParams = cohBean.getCoherenceClusterParams()
. . .
cohCParams.setClusteringMode('unicast')
cohCParams.setSecurityFrameworkEnabled('false')
cohCParams.setUnicastListenAddress( '192.168.0.100')
cohCParams.setUnicastListenPort( 8088)
cohCParams.setUnicastPortAutoAdjust( 'true')
. . .
```

Obtain an instance of the Coherence cluster Mbean.

See documentation for a complete list of properties.

Using the `setXYZ` methods, modify properties as required.

Parameters are set using a variety of `setXXX` methods on the Coherence clusters parameters object. There are several methods of the form `setXXX` based on the various parameters that can be set on a Coherence cluster. Alternatively, `WLST` provides a directory-style metaphor for accessing objects. To use this alternative style, change the directory to the Coherence cluster directory and set options as required, for example:

```
cd('CoherenceClusterSystemResource/myCoherenceCluster/CoherenceResou
rce/myCoherenceCluster/CoherenceClusterParams/NO_NAME_0')
set('ClusteringMode', 'unicast')
set('SecurityFrameworkEnabled','false')
set('UnicastListenAddress', '192.168.0.100')
set('UnicastListenPort', 8088)
set('UnicastPortAutoAdjust', 'true')
```

Note that WLST documentation has addition information and examples of setting other properties including logging, well-known addresses, and others.

Associating a Coherence Cluster with a WLS Cluster

To Create a WLS cluster and associate with a Coherence cluster:

```
cohClusterRes = create('clustername',
                       'CoherenceClusterSystemResource')

. . .

wlsClusterRes= create('wls cluster name', 'Cluster')

. . .

wlsClusterRes.setClusterMessagingMode('unicast')
wlsClusterRes.setCoherenceClusterSystemResource(cohClusterRes)
cohTier = wlsClusterRes.getCoherenceTier()
cohTier.setLocalStorageEnabled(false)

. . .
```

Associated the Coherence cluster with the WLS cluster.

Ensure messaging modes match.

Set local storage as required.

To associate a WLS cluster with a Coherence cluster:

- Create the Coherence cluster normally.
- Create the WLS cluster normally.
- Modify or configure the WLS cluster as required (represented by the ellipses).
- Set the WLS clusters messaging mode to match the Coherence cluster.
- Obtain the Coherence tier system resource and set local storage true or false as required. Note that the default is true.

WLST: Creating Managed Coherence Servers

To create a machine:

```
wlsClusterRes= create('wls cluster name', 'Cluster')
```

Associated the server with a WebLogic cluster.

```
wlsServer= create('server name','Server')
wlsServer.setListenPort(8001)
wlsServer.setCluster(wlsClusterRes)
wlsServer.setMachine('localhost')
. . .
memberConfig=wlsServe.getCoherenceMemberConfig()
memberConfig.setLocalStorageEnabled('true')
memberConfig.setsetRackName( 'rackname')
memberConfig.setRoleName( 'rolename')
memberConfig.setSiteName('sitename')
memberConfig.setUnicastListenAddress('192.168.0.100')
memberConfig.setUnicastListenPort(8088)
memberConfig.setUnicastPortAutoAdjust('true')
```

Specify Coherence member configuration.

To create a WebLogic instance and associate it with a WebLogic cluster (which itself is associated with a Coherence cluster):

1. Create the associated WebLogic cluster resource.
2. Create a WebLogic server.
3. Associate the WebLogic server instance with a WebLogic cluster associated with a Coherence cluster.

When the instance has been created and associated with a server, the Coherence specific configuration can be modified by obtaining an instance of the Coherence Member Configuration and then setting one or more settings.

Coherence*Web: Overview

Coherence*Web:

- Is a plug-in that enhances WebLogic Server's session management implementation with a Coherence grid
- Provides a highly scalable session replication solution that can span WebLogic domains and clusters
- Reduces WebLogic Server's memory footprint
- Requires no changes to existing application code
- Allows session data to be shared across applications
- Can be enabled on an application basis

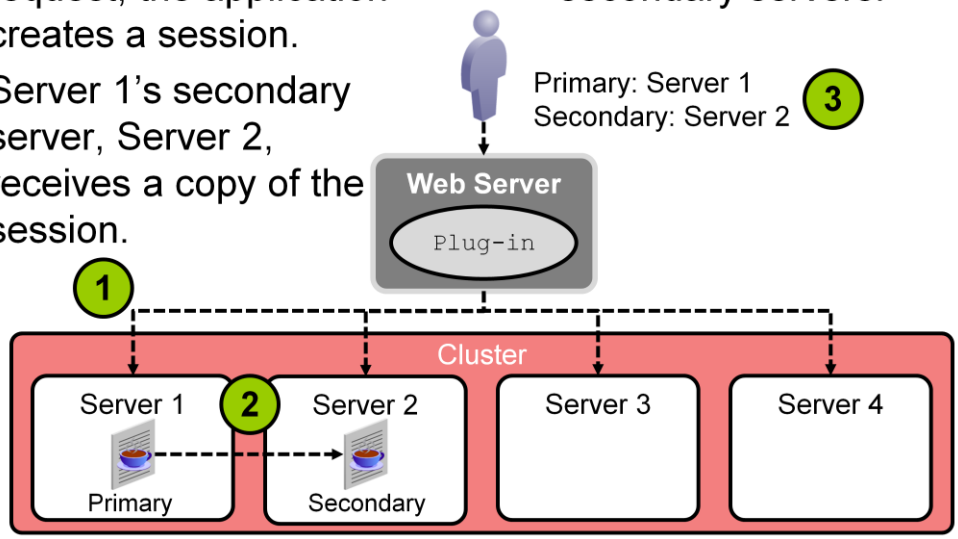
Coherence*Web is an application server plug-in dedicated to managing session state in clustered environments. It brings the scalability, availability, reliability, and performance characteristics of a Coherence data grid to in-memory session management and storage. Also, because it is not constrained by the deployment topologies of the application server, it enables session sharing and management across different web applications, domains, and even different application server products. Sometimes you may want to explicitly prevent session data from being shared by different Java EE applications that participate in the same Coherence cluster, so Coherence*Web supports this approach as well.

In the case where sessions are shared across web applications, you may want to scope individual session attributes so that they are either globally visible (that is, all web applications can see and modify these attributes) or scoped to an individual web application (that is, not visible to any instance of another application). The latter approach may be desired to help avoid namespace collisions when multiple applications use the same session attribute names.

With Coherence*Web, session data is stored outside of the application server, thereby freeing server heap space. This architecture also allows you to individually tune and scale the sizes of your application server clusters and session data grids.

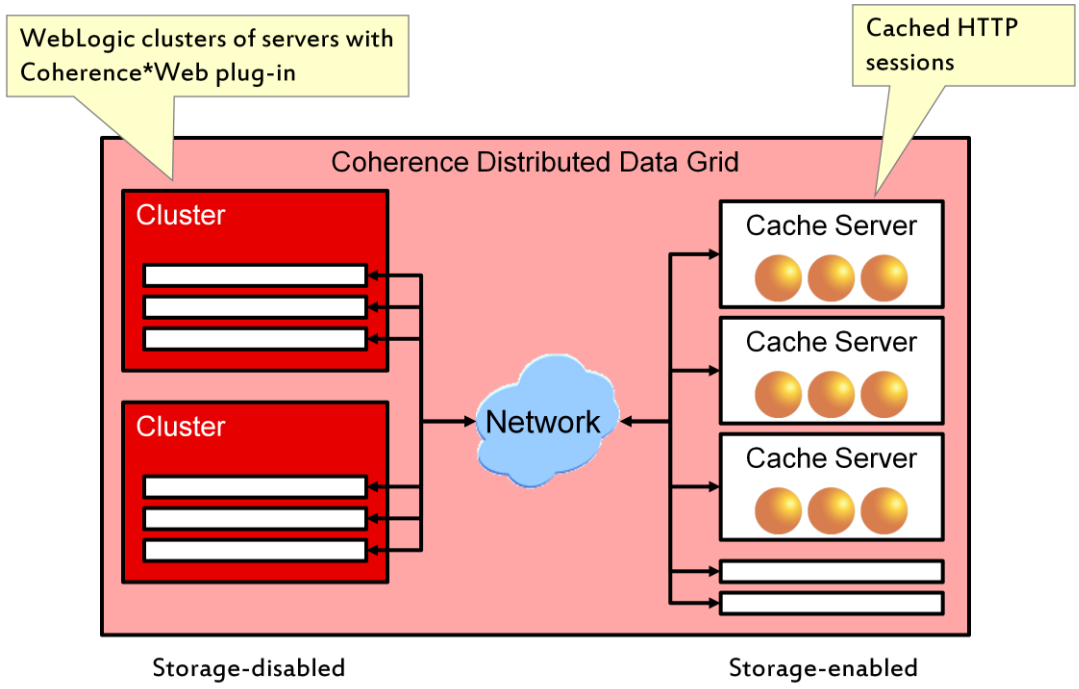
In-Memory Session Replication: Review

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). Subsequent requests from this client are routed (if possible) to the primary server, Server 1, which has the client's session information.

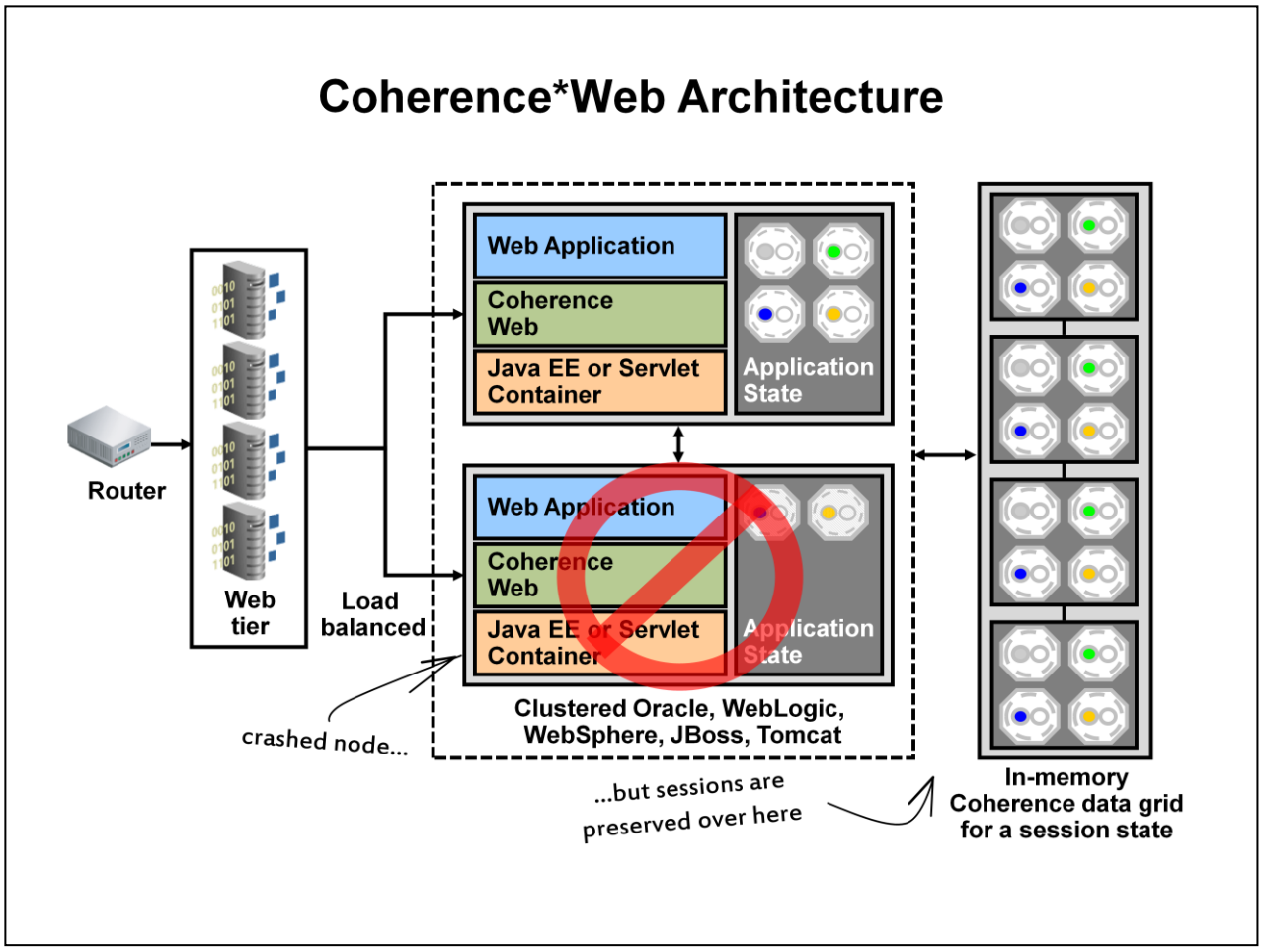
Coherence*Web and WebLogic Clusters



Coherence*Web is not a replacement for WebLogic Server's in-memory HTTP state replication services. However, Coherence*Web should be considered when an application has large HTTP session state objects, when running into memory constraints due to storing HTTP session object data, or if you have an existing Coherence cluster and want to off-load HTTP session storage to a Coherence cluster.

Coherence*Web is configured with local-storage disabled. This means that although the WebLogic Server instance is a member of the Coherence cluster, it is not a data storage member of the cluster. Storage-disabled members rely on other Coherence cluster members to manage and replicate data.

By default, Coherence*Web creates a single HTTP session across all web applications for each client and scopes the session attributes to each web application. This means that if a session is invalidated in one web application, that same session is invalidated for all web applications in WebLogic Server using Coherence*Web. If you do not want this behavior, a potential work-around is to edit the `<cookie-path>` element of the `weblogic.xml` descriptor.



The diagram in the slide illustrates that if a server were to fail, if you lose the entire application server or a JVM, Coherence still maintains the data. Therefore, you do not lose any session state information. Using your existing infrastructure, the user session fails over to one of the other application servers, or the one that is surviving. Therefore, no data is lost. The end user may not even notice that there is an outage. The user session continues without the user even noticing that anything went wrong because Coherence manages the data.

One common use case for Coherence clustering is to manage user sessions (conversational state) in the cluster. This capability is provided by the Coherence*Web module, which is a built-in feature of Oracle Coherence. Coherence*Web provides linear scalability for HTTP session management in clusters of hundreds of production servers. It can achieve this linear scalability because, at its core, it is built on Oracle Coherence dynamic partitioning.

Session management highlights the scalability problem that typifies shared data sources: If an application cannot share data across the servers, it would have to delegate that data management entirely to the shared store, which is typically the application's database. If the HTTP session is stored in the database, each HTTP request (in the absence of sticky load-balancing) would require a read from the database, causing the desired reads-per-second from the database to increase linearly with the size of the server cluster.

Coherence*Web Configuration

1. Start one or more Coherence cache servers on your network.
2. Update each application's `weblogic.xml` descriptor file and set the session persistence type to `coherence-web`.

```
<weblogic-web-app>
  <session-descriptor>
    <persistence-store-type>
      coherence-web
    </persistent-store-type>
  </session-descriptor>
</weblogic-web-app>
```

weblogic.xml

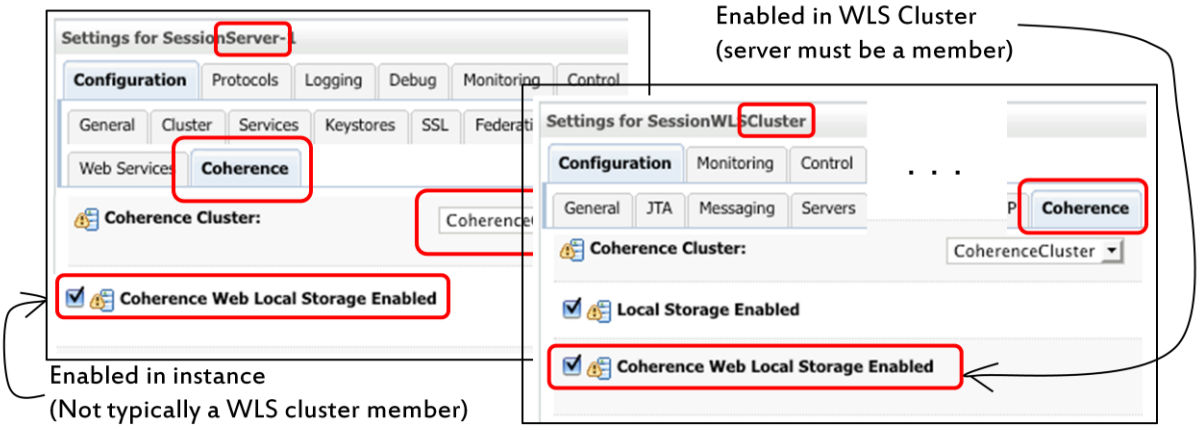
The WebLogic Server Coherence*Web plug-in consists of the `coherence-web.jar` and `coherence.jar` files, located in the `coherence/lib` directory in the Coherence distribution. When WebLogic is installed, these files are included in the WebLogic system classpath. The `coherence-web.jar` file will load application classes from the appropriate WebLogic classloader so you do not have to include Coherence JAR files in the web application's classpath. All Coherence*Web enabled applications running on WebLogic 12*c* (12.1.2) have application server scope. As a result, all applications become part of the same Coherence node.

The Coherence caches used by Coherence*Web are configured by the `default-session-cache-config.xml` file included within the `coherence-web.jar` file. Although the default configuration should be sufficient for most deployments, it can be overridden by placing a `session-cache-config.xml` file in your web application's `WEB-INF/classes` folder. If you are using the default session cache configuration file, then you can package and deploy your applications just like any other Java EE application. However, if you are using a custom session cache configuration file, then you must package and deploy your application in a GAR file.

Managed Coherence Server Coherence*Web Instance

Coherence*Web instances must:

- Be a member of a Coherence Cluster
- Must be enabled for Coherence*Web (member or WLS Cluster)



To support Coherence*Web enabled applications one or more Managed Coherence Servers must exist which are Coherence*Web enabled. A Managed Coherence Server may be Coherence*Web enabled in either of two ways:

- WebLogic Cluster – Enable all Managed Coherence Servers in a cluster by checking the Coherence Web Local Storage Enabled check box. Using this method all Managed Coherence Servers which are a member of this WebLogic Cluster support Coherence*Web session management.
- Managed Coherence Server – Enable specific Managed Coherence Server instances by selecting the instance and checking the Coherence*Web Local Storage Enabled checkbox.

Note that in either case the member may also be local storage enabled and the member or cluster must be part of a Coherence cluster.

When the Managed Coherence Server starts, two additional services are started to support session management, the DistributedSessions and ReplicatedSessionsMisc services. Both of which may be logged as starting in the log file. For example: