BEAWebLogic Portal ™

Deploying Portal Applications

Contents

Deploying Portal Applications

Deploying Portal Applications

This document covers the intricacies of deploying your WebLogic Portal application into a production environment. There are a number of different options for configuring your production portal application and domain. This document outlines some of your options but focuses primarily on the best practices for production environments, including using a enterprise-quality database and a clustered environment for redundancy and scalability.

For details on setting up a development environment and shared development domain, see the *Team Development Guide* at http://edocs.bea.com/wlp/docs81/teamdev/index.html.

This document contains the following sections:

- Preparing Your Portal Application

- Configuring a Portal Cluster

- Deploying a Portal Application to the Cluster

- Understanding Portal Resources

- Zero Downtime Architectures

Preparing Your Portal Application

To bring your portal online in a production environment, it is first necessary to prepare your portal application. Typical preparation steps include modifying deployment descriptors for product, building the enterprise archive (EAR) with all its pre-compiled classes, and determining if you want to compress that EAR into an archive or leave it exploded.

Configuring Portal Application Deployment Descriptors

Similar to any J2EE application, a portal application has a number of deployment descriptors that you may want to tune for your production environment.

Application Deployment Descriptors

Within the Portal application is the `/META-INF` directory which contains a number of deployment descriptors, including `application-config.xml`, a portal-specific deployment descriptor that contains cache configuration, behavior tracking, campaign, and commerce tax settings. If these values are different for your production environment than for your existing development settings, modify this file appropriately before building the portal application.

Web Application Deployment Descriptors

Within any portal Web application is a `/WEB-INF` directory that contains a number of deployment descriptors you may need to modify for your production environment.

- **`web.xml`** is a J2EE standard deployment descriptor. Among other settings, it has a set of elements for configuring security for the Web application. You can read more about web.xml here: http://edocs.bea.com/wls/docs81/webapp/web_xml.html.

- **`weblogic.xml`** is a standard WebLogic deployment descriptor for Web applications that has a number of important descriptor entries. Detailed information on this file can be found here: http://e-docs.bea.com/wls/docs81/webapp/weblogic_xml.html.

  Note: In a clustered production environment, it is important that you configure the `<session-param>` descriptor element in `weblogic.xml` to enable session replication to take place across the cluster. Without this setting, you will not have failover of a user's state information if a server in the cluster is stopped. You may need to add the following block to `weblogic.xml`.

```
<session-descriptor>
<session-param>
<param-name>PersistentStoreType</param-name>
```

```
<param-value>replicated_if_clustered</param-value>
```

```
</session-param>
```

```
</session-descriptor>
```

By default if PersistentStoreType is not set, it defaults to disabling persistent session storage.

The other commonly modified element in `weblogic.xml` for production environments is the `<jsp-descriptor>`. For production, it is common modifications include:

- Turn off debugging by setting debug to false.

- Precompile the JSPs in the Web application to reduce the time needed to display pages on their first invocation by setting precompile to true.

  Also set precompileContinue to true, because if any JSPs do not compile, deployment of the Web application stops.

  You may need to add these elements to `weblogic.xml` inside the existing `<jsp-descriptor>` section. For example:

```
<jsp-param>
<param-name>precompile</param-name>
<param-value>true</param-value>
<param-name>precompileContinue</param-name>
<param-value>true</param-value>
</jsp-param>
```

- In a compressed EAR environment, set pageCheckSeconds to -1 to disable polling of JSP pages for changes.

Workshop Deployment Descriptors

WebLogic Workshop has a number of additional deployment descriptors that are important if you are developing Web services. Information on these can be found on dev2dev in the *WebLogic Workshop Internals* document under "Application Customization" at:
http://dev2dev.bea.com/products/wlworkshop81/articles/wlw_internals.jsp#9.

Compile with Your Runtime JVM

If you are going to use a particular Java Virtual Machine (JVM) in your production environment, it is a good idea to compile the EAR application with the JDK for that JVM. You can change the JVM for your WebLogic Workshop project by going to **Tools > Application Properties**,

selecting **WebLogic Server**, and specifying the path to the JDK Home (root directory) you want to use.

Building a Portal Application with WebLogic Workshop

To deploy a portal application to a production environment, you must first build the application in WebLogic Workshop to compile necessary classes in the portal application. There are two options:

- **Build > Build EAR** – Most common. Use this option to compile the classes and build the portal application as a compressed EAR.

  or

- **Build > Build Application** – Use this option to compile the classes and build the portal application as an exploded (uncompressed) EAR. This option, however, many not invoke all the same processing as Build EAR. Review how Workshop treats deployment descriptors when Build EAR is run and determine if there is an impact. See the WebLogic Workshop Internals document at
  http://dev2dev.bea.com/products/wlworkshop81/articles/wlw_internals.jsp.

  Alternatively, to generate an uncompressed EAR, you can run **Build > Build EAR** and then uncompress the generated EAR file.

  After you create an exploded EAR, rename the application directory so that it has a `.ear` extension. For example, if the application directory is called `myPortalApp`, rename the directory to `myPortalApp.ear`. Adding the `.ear` extension to the directory name prevents deployment exceptions for exploded EARs.

Building In the Command Line

You can build your portal application from the command line using the wlwBuild command. This can make it easier for you to automate the process of building your application. See
http://e-docs.bea.com/workshop/docs81/doc/en/workshop/reference/commands/cmdWlwBuild.html.

Configuring a Portal Cluster

This section provides the steps necessary to set up a cluster across which your portal application is deployed.

Set up a Production Database

To deploy a portal application into production, it is necessary to set up an enterprise-quality database instance. PointBase is supported only for the design, development, and verification of applications. It is not supported for production server deployment.

Details on configuring your production database can be found in the *Database Administration Guide* at http://edocs.bea.com/wlp/docs81/db/index.html.

Once you have configured your enterprise database instance, it is possible to install the required database DDL and DML from the command line as described in the *Database Administration Guide*. A simpler option is to create the DDL and DML from the domain Configuration Wizard when configuring your production environment, as this guide will show.

Reading the wlw-manifest.xml File

When configuring your production servers or cluster with the domain Configuration Wizard, you will need to deploy some JMS queues that are required by WebLogic Workshop-generated components that are deployed at run time. To find the JMS queue names you need, open the `wlw-manifest.xml` file in the portal application's `/META-INF` directory.

In the file, find the JMS queue JNDI names that are the defined values in elements named `<con:async-request-queue>` and `<con:async-request-error-queue>`. Record the JNDI names of the JMS queues found in those definitions for use when configuring your production system.

You may also need to configure other settings in `wlw-manifest.xml`. For more information, see "How Do I: Deploy a WebLogic Workshop Application to a Production Server?" at http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/howdoi/howDeployWebLogic WorkshopWebServicestoaProductionServer.html.

Choosing a Cluster Architecture

By clustering a portal application, you can attain high-availability and scalability for that application. Use this section to help you choose which cluster configuration you want to use.

Single Cluster

When setting up an environment to support a production instance of a portal application, the most common configuration is to use the WebLogic Recommended Basic Architecture documented here: http://e-docs.bea.com/wls/docs81/cluster/planning.html#1090621.

Figure 1 shows a WebLogic Portal-specific version of the recommended basic architecture.

Figure 1   WebLogic Portal single cluster architecture
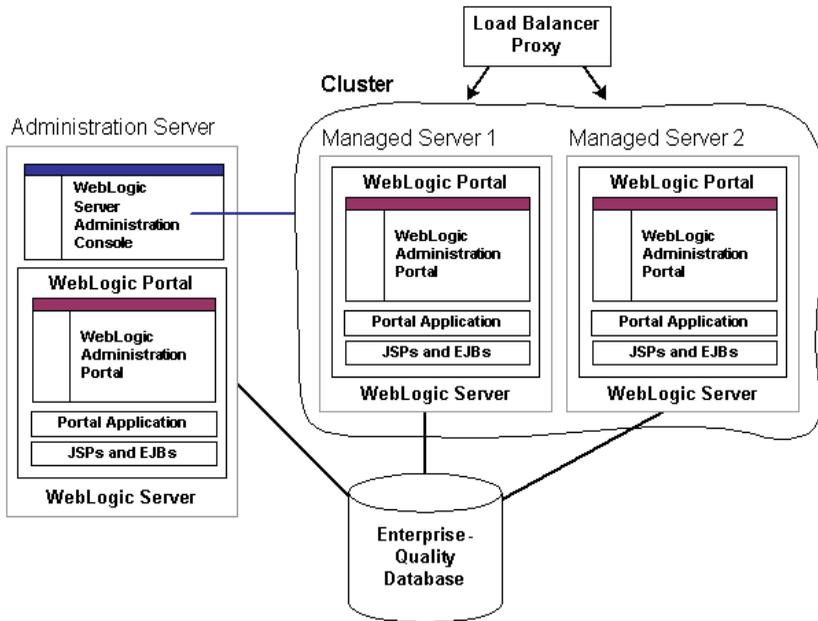
Note: WebLogic Portal does not support a split configuration architecture where EJBs and JSPs are split onto different servers in a cluster. The basic architecture provides significant performance advantages over a split configuration for Portal.

Even if you will be running a single server instance in your initial production deployment, this architecture allows you to easily configure new server instances if and when needed.

Multi Cluster

A multi-clustered architecture can be used to support a zero-downtime environment when your portal application needs to be accessible 365x24. While a portal application can run indefinitely in a single cluster environment, deploying new components to that cluster or server will result is some period of time where the portal is inaccessible. This is due to the fact that while a new EAR application is being deployed to a WebLogic Server, HTTP requests cannot be handled. Redeployment of a portal application also results in the loss of existing sessions.

A multi-cluster environment involves setting up two clusters, typically a primary cluster and secondary cluster. During normal operations, all traffic is directed to the primary cluster. When some new components (such as portlets) need to be deployed, the secondary cluster is used to handle requests while the primary is updated. The process for managing and updating a multi-clustered environment is more complex than with a single cluster and is addressed in "Zero Downtime Architectures" on page 50. If this environment is of interest you may want to review that section now.

Figure 2   Weblogic Portal multi-cluster architecture
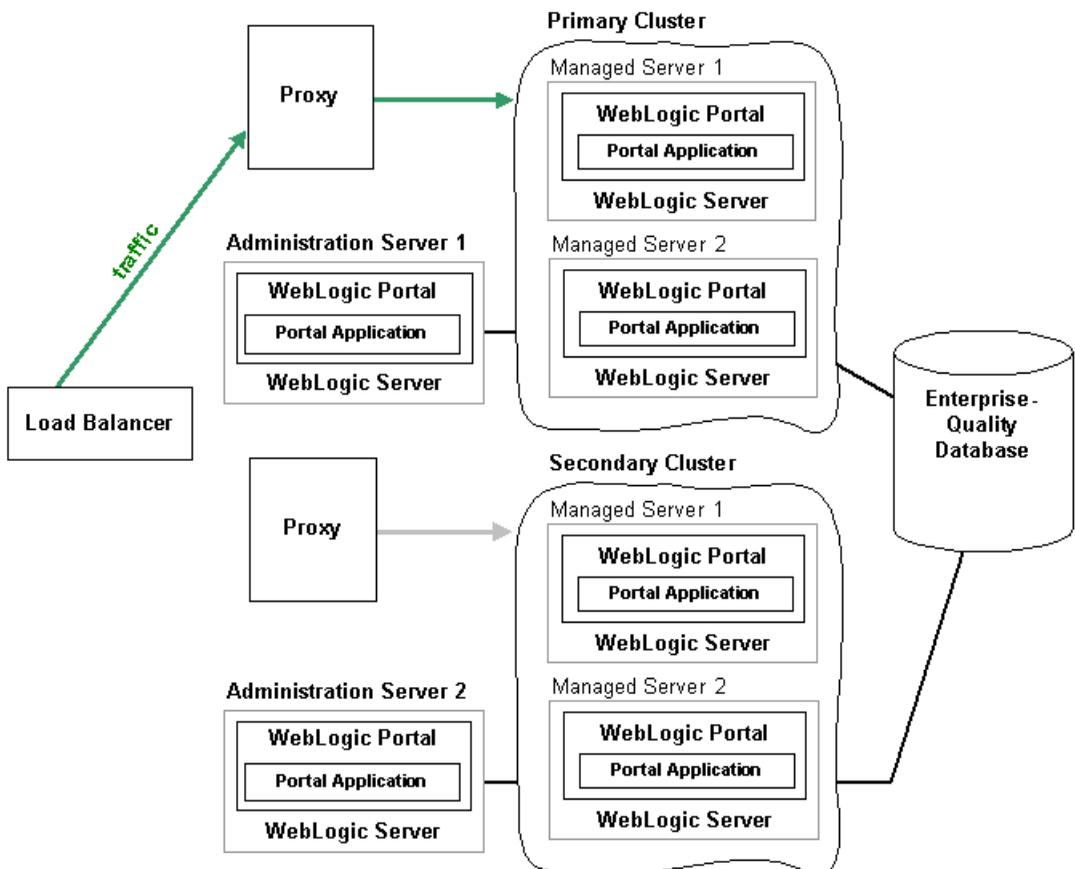
Configuring a Domain

You should determine the network layout of your domain before building your domain with the Configuration Wizard. Determine the number of managed servers you will have in your cluster—the machines they will run on, their listen ports, and their DNS addresses. Decide if you will use WebLogic Node Manager to start the servers. For information on Node Manager, see *Configuring and Managing WebLogic Server* at
http://e-docs.bea.com/wls/docs81/adminguide/nodemgr.html.

WebLogic Portal must be installed on the cluster's administration server machine and on all managed server machines.

Using the Configuration Wizard

Create your new production environment with the domain Configuration Wizard. See *Creating WebLogic Configurations Using the Configuration Wizard* at
http://edocs.bea.com/platform/docs81/confgwiz/intro.html.

Creating a Production Cluster Environment with the Configuration Wizard

This section walks you through the creation of a production cluster environment for WebLogic Portal.

In addition, you can see a demo of how a production environment is configured.

1. Start the Configuration Wizard. In Windows, choose **Start >Programs > BEA WebLogic Platform 8.1 > Configuration Wizard**.

2. In the Create or Extend a Configuration window, select **Create a new WebLogic configuration** and click Next.

3. In the Select a Configuration Template window, select **Basic WebLogic Portal Domain** and click Next.

4. In the Choose Express or Custom Configuration window, select **Custom** and click Next.

5. In the Configure the Administration Server window:

   a. Enter a name for your administration server.

   b. In the Listen Address field, leave the default "All Local Addresses" selection.

   c. Enter the listen port.

d. If you want to use the Secure Sockets Layer (SSL) protocol for secure access to portal application resources, select the SSL enabled option and enter an SSL listen port.

e. Click Next.

6. In the Managed Servers, Clusters, and Machines Options window, select **Yes** to customize the configuration settings, and click Next.

7. In the Configure Managed Servers window, add your managed servers. The number of managed servers you want in your cluster(s) will vary depending on your choice of hardware.

   In the Listen Address field, enter the IP address for each managed server. Do not leave the default "All Local Addresses" setting.

   When you are finished adding managed servers, click Next.

8. In the Configure Clusters window, add your cluster(s). Choose a multicast address that is not currently in use. Choose the Cluster addresses for the managed servers in this cluster. These take the form of a comma-separated list of the DNS alias names of the managed servers.

   See "Cluster Address" in *Using WebLogic Server Clusters* at http://edocs.bea.com/wls/docs81/cluster/setup.html#ClusterAddress for more information.

   When you are finished, click Next.

9. In the Assign to Clusters window, choose all the managed servers you want to associate with each cluster by moving the server names from the left pane to the right pane. Click Next.

10. In the Configure Machines window, you can create logical representations of the systems that host your server instances. Host information is used for locality routing, especially during session replication. If you are running more than one managed server per machine, it is important that you configure host information so that WebLogic server does not replicate a session on the same machine.

    Also, if you will be using Node Manager to manage starting and stopping your servers, you should specify that information here.

    Click Next.

11. If you choose to Configure Machines, in the Assign Servers to Machines window, target the servers to the appropriate machines.

12. In the Database (JDBC) Options window, select **Yes** to define JDBC components, and click Next.

13. In the Configure JDBC Connection Pools window, there will be a cgPool tab. Change the cgPool Vendor to use your production database type, and then specify the information needed to connect to that database instance such as the host and port information.

    Make the same changes on the cgJMSPool-nonXA and portalPool tabs.

    For cgJMSPool-nonXA, in the Driver field, be sure to select the non-XA driver.

    Click Next.

14. In the Configure JDBC Multipools window, click Next.

15. In the Configure JDBC Data Sources window, you should see a list of JDBC Data Sources configured. Click Next.

16. In the Test JDBC Connection Pool and Setup JDBC Database window, select **cgPool** in the Available JDBC Connection Pools pane and click **Test Connection**.

    If you have not already created the database objects for the portal application in your database instance, select your database version in the **DB Version** field and click Load Database.

    Warning: Exercise caution when loading the database, as the scripts will delete any portal database objects from the database instance if they already exist. You will see a large number of SQL statements being executed. It is normal for the scripts to have a number of statements with errors on execution, because the script drops objects that may not have been created before.

    Click Next.

17. In the Messaging (JMS) Options window, select **Yes** to define JMS components, and click Next.

18. In the Configure JMS Connection Factories window, you should see the cgQueue. Its Default delivery mode should be set to Persistent. Click Next.

19. In the Configure JMS Destination Key(s) window, click Next.

20. In the Configure JMS Template(s) window, click Next.

21. In the Configure JMS File Stores window, validate that FileStore exists, and click Next.

22. In the Configure JMS JDBC Store window, validate that you have JMS stores for the administration server and for each of the managed servers, typically named cgJMSStore_auto_1, cgJMSStore_auto_2, and so on. To create a JMS store:

    a. Click Add.

    b. Specify a name, such as `cgJMSStore_auto_a` for the administration server.

    c. In the Connection Pool field, select the same connection pool used for all stores (such as cgJMSPool-nonXA).

    d. In the Prefix Name field, specify a unique JMS prefix name (such as `por_a` for the administration server).

       If you are using Oracle as a database, specify the Oracle schema name before the prefix name. For example, `OSCHEMA.por_a`.

    e. Click Next.

23. In the Configure JMS Servers window, validate that you have servers (including the administration server) that correspond to the JMS stores created, typically named cgJMSServer_auto_1, cgJMSServer_auto_2, and so on. To create a JMS server:

    a. Click Add.

    b. Specify a name, such as `cgJMSServer_auto_a` for the administration server.

    c. In the Store field, select the corresponding JMS JDBC store you created in the previous window.

    d. Click Next.

24. In the Assign JMS Servers to WebLogic Servers window, assign the JMS Servers to the administration server and to each of the managed servers. Click Next.

25. In the Configure JMS Topics window, click Next.

26. In the Configure JMS Queues window, click Next.

27. In the Configure JMS Distributed Topics window, click Next.

28. In the Configure JMS Distributed Queues window, you need to add new queues that are required by WebLogic Workshop. The JNDI names for these queues can be found in your application's `/META-INF/wlw-manifest.xml` file, as described in .

These names will be something like `<WEB_APP>.queue.AsyncDispacher` and `<WEB_APP>.queue.AsyncDispacher_error`. For each queue, add a new JMS Distributed Queue with the Add button. Set the Name entry and JNDI name entry to the name listed in `wlw-manifest.xml`. Set the Load balancing policy and Forward delay as appropriate for your application.

A pair of queue entries exists for each Web application (portal Web project) in your portal application. When you are finished, you should have a distributed queue for each queue. In other words, if your enterprise application has three Web applications, you should have added six distributed queues—two for each Web application.

You do not need to create queues for the WebLogic Administration Portal Web application.

Note: If you are using multiple clusters, create an additional set of queues for each cluster. For example, if you have a Web application called basicWebApp, and you are using a second cluster, create a unique basicWebApp queue for that cluster named something like `basicWebApp.queue.AsyncDispatcher.2`. When you do this, the Configuration Wizard does not let you enter the same JNDI name for multiple queues. In this example, enter a JNDI name that ends with a ".2". Later in the setup procedures you will make all JNDI names the same in the WebLogic Administration Console for each Web application.

Click Next.

29. In the Assign JMS Distributed Destinations to Servers or Clusters window, target your newly defined queues to your cluster. In the right pane, select the cluster, in the left pane, select the queue(s), and click the right arrow icon. Click Next.

30. In the JMS Distributed Queue Members window, click Next. You will create distributed JMS queue members in "Setting up JMS Servers" on page 14.

31. In the Applications and Services Targeting Options window, select **Yes** and click Next.

32. In the Target Applications to Servers or Clusters window, target all applications to the administration server as well as to the cluster. Click Next.

33. In the Target Services to Servers or Clusters window, click Next.

34. In the Configure Administrative Username and Password window, enter a username and password for starting the administration server. You do not need to configure additional users, groups, and global roles, so make sure **No** is selected at the bottom of the window. Click Next.

35. If you are installing on Windows, in the Configure Windows Options window, select the options you want for adding a shortcut to the Windows menu and installing the administration server as Windows service. The wizard always creates a single default shortcut on the windows menu regardless of what you select for the Create Start Menu option. The option lets you create an additional shortcut with different settings.

    If you chose to create a Windows menu shortcut for your domain, click Next in the Build Start Menu Entries window.

    Click Next.

    For information on starting WebLogic Server, see "Creating Startup Scripts" at http://e-docs.bea.com/wls/docs81/isv/startup.html.

36. In the Configure Server Start Mode and Java SDK window, select **Production Mode** and select the SDK (JDK) you want to use. Click Next.

37. In the Create WebLogic Configuration window, browse to the directory where you want to install your administration server domain and enter a Configuration Name.

    To avoid path length exceptions, use a short path for the domain, such as `<drive>:/ourDomain`.

38. Click Create.

39. When the domain is created, click Done.

Configuring the Administration Server

At this point your administration server domain has been configured using the domain Configuration Wizard. Before you start the administration server to do additional configuration work, you may want to perform one or both of the following setup tasks: increasing the default memory size allocated to the administration server and allowing server startup without requiring authentication by creating a `boot.properties` file.

Increasing the Default Memory Size

To increase the default memory size allocated to the administration server, you will need to modify your `startWebLogic` script in the domain's root directory and change the memory arguments. For example:

In Windows, change:

`set MEM_ARGS=-Xms256m %memmax%` to `set MEM_ARGS=-Xms512m -Xmx512m`

In Unix, change:

```
MEM_ARGS="-Xms256m ${memmax}" to MEM_ARGS="-Xms512m -Xmx512m"
```

The exact amount of memory you should allocate to the server will vary based on a number of factors such as the size of your portal application, but in general 512 megabytes of memory is recommended as a default.

Allowing Server Startup Without Requiring Authentication

To allow server startup without requiring authentication, create a `boot.properties` file in your domain root directory that contains the username and password you want to log in with. For example:

```
username=weblogic
password=weblogic
```

After the server starts for the first time using this file, the username and password are encrypted in the file.

Setting up JMS Servers

In this procedure you will finish configuration of the JMS servers you began to configure in the Configuration Wizard.

1. Start the administration server and log in to the WebLogic Server Administration Console, found at http://<server>:<port>/console.

2. Configure the JMS distributed destinations.

   a. Expand **Services > JMS > Distributed Destinations**. Perform the following sub-steps for each queue you defined earlier for the WebLogic Workshop components.

      Note: You do not need to configure the `dist_cgJWSQueue_auto` queue.

   b. Select the queue name and select the **Auto Deploy** tab.

   c. Click **Create members on the selected Servers (and JMS Servers)**.

   d. Select your cluster(s) to target the JMS queue to, and click Next.

   e. Select all the managed servers in the cluster to create members for the queue, and click Next.

   f. Select all the JMS Servers where members will be created, and click Next.

   g. Commit the changes by clicking Apply.

h. If you are using multiple clusters, change the JNDI names you assigned to the cluster queues in Creating a Production Cluster Environment with the Configuration Wizard step 28. Select each cluster queue (for example, select `basicWebApp.queue.AsyncDispatcher.2`), click the General tab, and remove the ".2" suffix (or whatever unique identifier you used in the Configuration Wizard) so that all JNDI names are the same for each Web application. For example, all JNDI names for the basicWebApp should be `basicWebApp.queue.AsyncDispatcher`. After each change, click Apply.

3. Configure the JMS servers for the managed servers.

   a. Expand **Services > JMS > Servers >** *managed server JMS server name* **> Destinations**. Perform the following sub-steps for each member queue ending in `AsyncDispatcher` (but not for member queues ending in `AsyncDispatcher_error`).

   b. Select the queue, and select the Redelivery tab.

   c. In the Error Destination field, select the companion error queue for the queue.

   d. Click Apply.

4. Create JMS queues for the administration server.

   a. Select **Services > JMS > Servers >** *administration JMS server name* **> Destinations**.

   b. Click Configure a New JMS Queue.

   c. Create JMS queues and error queues for each Web application. Use any name, but for convention you can make the Name the same as the JNDI Name.

      View your application's `META-INF/wlw-manifest.xml` file to see the queues you must create. See "Reading the wlw-manifest.xml File" on page 5.

      For example, if you have two Web applications, basicWebApp and bigWebApp, create the following queues for the administration server JMS server:

```
basicWebApp.queue.AsyncDispatcher
basicWebApp.queue.AsyncDispatcher_error
bigWebApp.queue.AsyncDispatcher
bigWebApp.queue.AsyncDispatcher_error
```

      You must also create a single queue named `jws.queue` if it does not already exist.

      Click Create after you create each queue.

5. Retarget the JMS servers to "migratable" to support JMS failover.

   a. Expand **Services > JMS > Servers**. Perform the following sub-steps for each JMS server.

b. Select the JMS server, and select the Target and Deploy tab.

c. In the Target field, select the *target* (migratable) counterpart item for the previously selected target. For example, if the target was `managed1`, select `managed1 (migratable)`.

d. Click Apply.

Creating Managed Server Directories

Now that you have configured your domain, including defining your managed servers, you need to create a server root directory for each managed server. There are many options for this, depending on whether or not the managed server will reside on the same machine as the administration server and whether or not you will use the Node Manager.

- Most of the files in the domain-level directory are not necessary for managed servers, so a domain (files directly in the domain directory) is not required on each managed server, especially if you are using the Node Manager to start and stop managed servers. For example, `config.xml` in a managed server domain is not used. Instead, the `config.xml` file in the administration server is used. The only requirement for managed servers is to have the `wsrpKeystore.jks` file one directory above the server directory (in the equivalent of a domain-level directory).

- If the managed server will run on a different machine than the administration server and you will not use Node Manager, the easiest option is to use the Configuration Wizard to create a full file system domain for the managed server, as described in the following procedure.

WebLogic Portal must be installed on all managed servers.

In addition, you can see a demo of how to create a managed server.

1. To create a new managed server, launch the Configuration Wizard.

2. Choose to **Create a new WebLogic configuration** and click Next.

3. In the Select a Configuration Template window, select Basic WebLogic Portal Domain and click Next.

4. In the Choose Express or Custom Configuration window, select **Express** and click Next.

5. In the Configure Administrative Username and Password window, enter a username and password for the server and click Next.

This information will not typically be used, because you will bind this server to the administration server using the administration server's credentials.

6. In the Configure Server Start Mode and Java JDK, select **Production Mode** and the SDK (JDK) you will use with the domain. Click Next.

   It is important you choose the same JDK across all instances in the cluster.

7. In the Create WebLogic Configuration window, choose the directory you want to install to, and in the **Configuration Name** field, enter a domain name to use. For best practices, choose a domain name like 'managedServer1', 'managedServer2', and so on.

8. Click Create.

9. When the domain is created, click Done.

10. If you want to allow server startup without requiring authentication on each managed server, create a `boot.properties` file in each managed server's domain directory (or one level above the server directory) that contains a username and password. For example:

```
username=weblogic
password=weblogic
```

   After the initial server startup using `boot.properties`, the username and password are encrypted in the file.

Once you have created a file system domain directory for a managed server, you can reuse the same domain for your other managed server on the same machine by specifying different servername parameters to your startManagedWebLogic script, or create new managed domains using the domain Configuration Wizard.

Note: If you decide not to use a full domain for your managed servers (that is, not include all files in the domain-level directory), be sure you keep or put a copy of `wsrpKeystore.jks` in the directory directly above the server directory (in the equivalent of the domain-level directory).

Increasing the Default Memory Size

To increase the default memory size allocated to a managed server (if you are not using Node Manager), you will need to modify your `startManagedWebLogic` script in the managed server root directory and change the memory arguments. For example:

In Windows, change:

```
set MEM_ARGS=-Xms256m %memmax% to set MEM_ARGS=-Xms512m -Xmx512m
```

In Unix, change:

```
MEM_ARGS="-Xms256m ${memmax}" to MEM_ARGS="-Xms512m -Xmx512m"
```

The exact amount of memory you should allocate to the server will vary based on a number of factors such as the size of your portal application, but in general 512 megabytes of memory is recommended as a default.

Deploy a New Application

This section provides instructions for the initial deployment of your portal application.

In addition, you can see a demo of how to deploy a portal application.

At this point we can add deploy your portal application to the cluster. First, you should place the .ear file (or exploded EAR) on the file system of the administration server. To make it easier to redeploy changes to the application, place the file in a known location from which you will always deploy the application, such as the root directory of the administration domain.

The steps for initial application deployment to the administration server and managed servers, and the order with which you start the managed servers, differ in different scenarios. Table 1 describes different scenarios and the deployment/startup procedures that must be used for each.

Table 1 Initial deployment/startup scenarios

Scenario	Deployment/Startup Procedures
Application not previously deployed, only administration server running – application does not use commerce (no `commerce*.jar` files in the application root directory)	1. Make sure the EAR (compressed or exploded) is in the location you want on the administration server. 2. Start the administration server. 3. In the WebLogic Server Administration Console (http://<adminserver>:<port>/console), select **Deployments > Applications**. 4. Click **Deploy a new Application**, and select the archive for the application from the file system. Click **Target Application**. 5. Target the application to the administration server and the cluster. 6. Deploy the application. 7. Once deployment has completed, start the managed servers however you like, such as in parallel (simultaneously).

Table 1 Initial deployment/startup scenarios (Continued)

Scenario	Deployment/Startup Procedures
Application not previously deployed, only administration server running – application uses commerce (has the `commerce*.jar` files in the application root directory)	Perform the previous deployment steps except the last step. When you start the managed servers, start a single managed server first. After this managed server has started, you can start the remaining managed servers in parallel.
Administration server and cluster up and running (whether or not commerce is being used)	1. Make sure the EAR (compressed or exploded) is in the location you want on the administration server. 2. In the WebLogic Server Administration Console on the administration server (http://<adminserver>:<port>/console), select **Deployments > Applications**. 3. Click **Deploy a new Application**, and select the archive for the application from the file system. Click **Target Application**. 4. Target the application to the administration server only. 5. Deploy the application on the administration server. 6. Once deployment has completed, select **Deployments > Applications > *appName***. 7. On the Targets tab, target the administration server and the cluster, and click Apply. 8. On the Deploy tab, redeploy the application.

Most components must be targeted to the administration server as well as the cluster. Here are the following exceptions:

– `<AppName>Admin` is not required on the administration server (though deploying `<AppName>Admin` on the administration server will not cause any problems).

– `<AppName>Datasync` is not required on the target server or cluster (though deploying `<AppName>Datasync` to the target server or cluster will not cause any problems).

This full level of deployment on the administration server and the cluster is required, and it is the only supported configuration. There are several application design challenges

specific to clustering that WebLogic Portal solves to ensure that portal applications perform properly and optimally in a cluster environment. The full targeting scheme described is part of the solution to those design challenges.

If you want to reduce the number of modules deployed on the administration server and the cluster, click Target Each Module in the deployment steps and untarget `<AppName>Admin` from the administration server and `<AppName>Datasync` from the cluster.

While you need to deploy your portal application to the administration server, the administration server is not typically used to serve pages for portal applications.

Using Targeted Deployment

In the previous steps, you targeted the entire application for deployment rather than targeting individual modules to the administration server and to the cluster. This is the recommended deployment approach. Since almost all modules must be deployed to both the administration server and to the cluster, any performance or disk space benefits you want to achieve by targeting individual modules is not significant.

However, if for some reason you want to use targeted deployment, use the following recommendations, which are listed in order from easiest to most difficult.

- Avoid targeted deployment. Deploy the entire application to the administration server and to the cluster.

- Use Netscape or Mozilla browsers to perform targeted deployment in the WebLogic Administration Console.

- Perform multiple targeted deployments. Deploy half of the modules first, then deploy the rest.

- Run the weblogic.Deployer utility in Linux or UNIX environments. For information on using the weblogic.Deployer utility, see http://e-docs.bea.com/wls/docs81/deployment/tools.html.

- Write Java code to call the `weblogic.Deployer main()` and pass in the module arguments.

- Use the WebLogic Server Scripting Tool (WLST). This option has not been tested on WebLogic Portal deployment. For information on WLST, see http://dev2dev.bea.com/codelibrary/code/wlst.jsp.

Starting Managed Servers

The sequence with which you start managed servers is important and depends upon your application deployment. See Table 1 for the startup sequences you must use.

If you are not deploying (a new application or an updated application), you can start all managed servers in parallel (simultaneously).

There are numerous ways to start a managed server and bind it to your administration server, including using Node Manager. For your initial setup, you may want to use the `startManagedWebLogic` script in the domain root directory. You can run this script by specifying the name of the managed server for this server instance and the URL of the administration server. Before starting the script, you should edit it and give the managed server more memory than it is allocated by default. This can be done by specifying a new `MEM_ARGS` setting. For example, change the memory allocation to `-Xms512m -Xmx512m`.

After starting a managed server, you can browse your portal application by going to the appropriate URL on the managed server instance. To provide your users a single point of entry to your cluster, as well as support session failover, you will need to configure a proxy server.

Configuring Your Proxy Server

For instructions on configuring a proxy plugin for WebLogic, see "Configure Proxy Plugins" in *Using WebLogic Server Clusters* at http://edocs.bea.com/wls/docs81/cluster/setup.html#684345.

There are no WebLogic Portal-specific configuration tasks when setting up a proxy plug-in.

Deploying a Portal Application to the Cluster

This section contains instructions for redeployment, partial redeployment, and iterative deployment of datasync data, such as user profile properties, user segments, content selectors, campaigns, discounts, and other property sets.

Redeploying Applications

Use the following procedure to redeploy an application.

1. In the WebLogic Server Administration Console on the administration server (http://<adminserver>:<port>/console), select **Deployments > Applications**.

2. In the list of deployed applications, click the trash can icon next to the application you want to redeploy. This undeploys the current version of the application.

3. After undeployment (when the application name disappears from the list of applications), replace the EAR with the updated application on the file system.

4. Redeploy the application, targeting the administration server and the cluster.

5. Run the Datasync Web application if you have updated datasync files. See "Iterative Datasync Deployment" on page 25.

Redeploying a Portal Application with weblogic.Deployer

You can use the WebLogic Server Administration Console or weblogic.Deployer tool to redeploy an updated portal application to your production server. See "weblogic.Deployer Utility" in *Deploying WebLogic Server Applications* at http://e-docs.bea.com/wls/docs81/deployment/tools.html.

The following batch file is an example of how to use weblogic.Deployer to redeploy a portal application to production.

```
@echo off
echo Redeploys a Portal Web Project to a Server or Cluster
echo First Parameter is the name of the Server or Cluster
echo Second Parameter is the name of the Application
echo Third Parameter is the administrative username for the Portal Server
set SERVER=%1
set APPNAME=%2
set USERNAME=%3
echo server = %SERVER%
echo appname = %APPNAME%
echo username = %USERNAME%
java weblogic.Deployer -redeploy -username %USERNAME% -name %APPNAME%
-targets %SERVER%
```

Partial Redeployment with weblogic.Deployer

In certain situations you can reduce the time needed to redeploy individual pieces of a portal application by using the weblogic.Deployer tool.

If your updates are contained within a particular portal Web application, you can redeploy just that Web application and greatly reduce the time spent in redeployment. This is of use if you have new portlets and Page Flows, but no new EJBs, libraries, or modules (which are enterprise application scoped).

Because a portal Web application has a number of dependencies on WebLogic Workshop control classes, those needed to be redeployed as well. The following batch file can be used to help simplify that process. You will need to have weblogic.Deployer in your classpath, which can be added by running `<BEA_HOME>/weblogic81/common/bin/commEnv` script.

```
@echo off

echo Redeploys a Portal Web Project to a Server or Cluster

echo First Parameter is the name of the Server or Cluster

echo Second Parameter is the name of the Application

echo Third Parameter is the name of the Portal Web Application

echo Fourth Parameter is the administrative username for the Portal Server

set SERVER=%1

set APPNAME=%2

set WEBAPPNAME=%3

set USERNAME=%4

echo server = %SERVER%

echo appname = %APPNAME%

echo webappname = %WEBAPPNAME%

echo username = %USERNAME%

set TARGETS=%APPNAME%@%SERVER%

set
TARGETS=%TARGETS%,.workshop/%APPNAME%/EJB/TimerControl_-livsjc6qp6ws@%SERV
ER%

set
TARGETS=%TARGETS%,.workshop/%APPNAME%/EJB/p13controls_k3cw9vg6497r@%SERVER
%
```

```
set
TARGETS=%TARGETS%,.workshop/%APPNAME%/EJB/MDBListener_-1x0154i4jz0he@%SERV
ER%

set TARGETS=%TARGETS%,.workshop/%APPNAME%/EJB/GenericStateless@%SERVER%

set TARGETS=%TARGETS%,.workshop/%APPNAME%/EJB/ProjectBeans@%SERVER%

java weblogic.Deployer -redeploy -username %USERNAME% -name %WEBAPPNAME%
-targets %TARGETS%
```

Iterative Datasync Deployment

This section provides instructions for updating portal application datasync data, such as user profile properties, user segments, content selectors, campaigns, discounts, and other property sets, which must be bootstrapped to the database in a separate deployment process.

Portal Datasync Definitions

Portal allows you to author a number of definition files, such as user profiles and content selectors, that must be managed carefully when moving from development to production and back.

Within WebLogic Workshop, portal definitions are created in a special Datasync Project, exposed in the WebLogic Workshop Application window as a `/data` subdirectory. (On the file system, the directory exists in the application's `/META-INF/data` directory.) This project can contain user profile property sets, user segments, content selectors, campaigns, discounts, catalog property sets, event property sets, and session and request property sets.

Datasync Definition Usage During Development

During development, all files created in the datasync project are stored in the META-INF/data directory of the portal application and exposed in WebLogic Workshop in the `<portalApplication>/data` directory. To provide speedy access from runtime components to the definitions, a datasync facility provides an in-memory cache of the files. This cache intelligently polls for changes to definitions, loads new contents into memory, and provides listener-based notification services when content changes, letting developers preview datasync functionality in the development environment.

Datasync definition modifications are made not only by WebLogic Workshop developers, but also by business users and portal administrators, who can modify user segments, campaigns, placeholders, and content selectors with the Weblogic Administration Portal. In the development

environment, both WebLogic Workshop and the WebLogic Administration Portal write to the files in the META-INF/data directory.

Compressed Versus Exploded EAR

When deployed into a production system, portal definitions often need to be modifiable using the Weblogic Administration Portal. In most production environments, the portal application will be deployed as a compressed EAR file, which limits the ability to write modifications to these files. In a production environment, all datasync assets must be loaded from the file system into the database so the application can be updated.

Figure 3 shows how the /data directory from the updated portal application is put into a standalone JAR and bootstrapped to the database.

Figure 3   Loading updated datasync files to the database
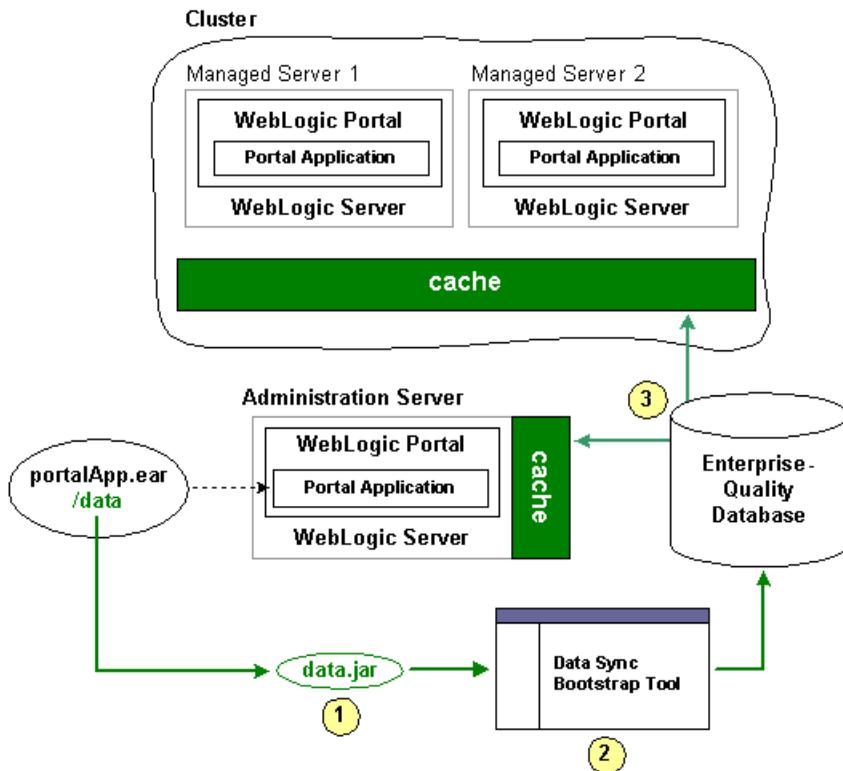
Alternatively, some production environments deploy their portal applications as exploded EARs. In this case, the deployed portal application on the administration server is the primary store of datasync definitions. Work done in the WebLogic Administration Portal on any managed server is automatically synchronized with the primary store.
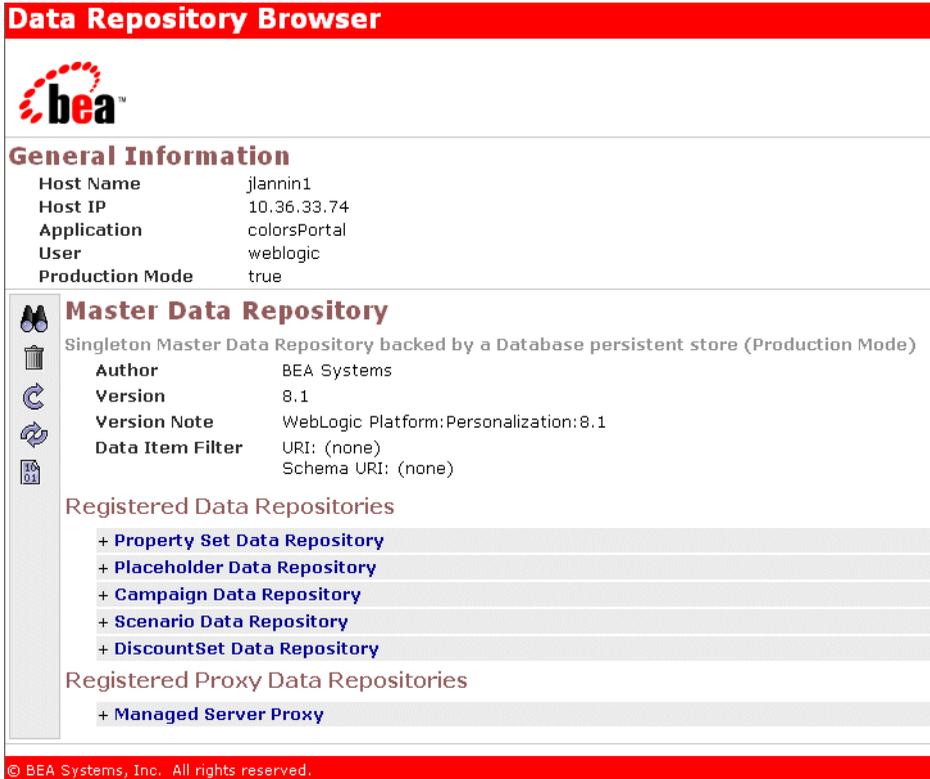
For both compressed and uncompressed EAR files, you can view and update datasync definitions using the Datasync Web Application.

Datasync Web Application

Each portal application contains a Datasync Web Application located in `datasync.war` in the application root directory. Typically, the URL to the Datasync Web application is http://<server>:<port>/<appName>DataSync. For example, http://localhost:7001/portalAppDataSync. You can also find the URL to your Datasync Web application by loading the WebLogic Server Administration Console and selecting **Deployments > Applications >** *appName* **> *DataSync** and clicking the **Testing** tab to view the URL.

The Datasync Web application allows you to view contents of the repository and upload new content, as shown in Figure 4.

Figure 4   Datasync Web application home page

Working with the Repository Browser – When working with the Data Repository Browser, you have the option to work with all the files in the repository using the icons on the left hand side of the page, or drill down into a particular sub-repository, such as the repository that contains all Property Set definitions.

View Contents – To view the contents of a repository, click on the binoculars icon to bring up the window shown in Figure 5.

Figure 5   Browsing the Datasync repository

From this list, click on a particular data item to see it's contents, as shown in Figure 6.

Figure 6   Data item contents

As you can see in the previous figure, you can view the XML data for a particular content item.

Removing content

To remove content from a repository, click on the trash can icon on the left side of the page.

Working with Compressed

When the application is deployed, if the JDBC Repository is empty (no data), then the files in the EAR will be used to bootstrap (initialize) the database. The Datasync assets are stored in the following tables: DATA_SYNC_APPLICATION, DATA_SYNC_ITEM, DATA_SYNC_SCHEMA_URI, and DATA_SYNC_VERSION. The bootstrap operation by

default only happens if the database is empty. When you want to do incremental updates, the Datasync Web application provides the ability to load new definitions directly into the database. This can be done as part of redeploying a portal application, or independently using a special JAR file that contains your definitions, as shown in Figure 6, "Data item contents," on page 30.

Upload new contents – In the Datasync Web application, there is a button on the left side that looks like a document with 1's and 0's called Bootstrap Data. When you click this icon, the following page appears, which lets you load data into the database.

Figure 7   Uploading new datasync data

When you bootstrap, you can choose a bootstrap source, which is either your deployed portal application or a stand-alone JAR file. For example, if you have an updated portal application that you have redeployed to your production environment, you can add any new definitions it contains to your portal. Alternatively, if you have authored new definitions that you want to load independently, you can create a JAR file with just those definitions and load them at any point.

Either way, when you update the data repository, you can choose to "Overwrite ALL data in the Master Data Repository," "Bootstrap only if the Mastery Repository is empty," or "Merge with Master Data Repository–latest timestamp wins."

Bootstrapping from an EAR – If you are redeploy an existing EAR application and want to load any new definitions into the database, choose the **Application Data (META-INF/data)** as your bootstrap source, and then choose the appropriate Bootstrap Mode. To ensure you do not lose any information, you may want to follow the instructions in the section entitled, "Pulling Definitions

to create a backup first. It is not possible to bootstrap definition data from an EAR file that is not deployed.

Creating a JAR file – To bootstrap new definition files independently of updates to your portal application, you can create a JAR file that is loaded onto the server that contains the files (content selectors, campaigns, user segments, and so on) that you want to add to the production system.

To do this, you can use the jar command from your `META-INF/data` directory. For example:

```
jar -cvf myfiles.jar *
```

This example will create a JAR file called `myfiles.jar` that contains all the files in your data directory, in the root of the JAR file. Then, you can bootstrap information from this JAR file by choosing **Jar File on Server** as your data source, specifying the full physical path to the JAR file and choosing the appropriate bootstrap mode. By running this process you can upgrade all the files that are packaged in your JAR. Controlling the contents of your JAR allows you to be selective in what pieces you want to update.

When creating the JAR file, the contents of the `META-INF/data` directory should be in the root of the jar file. Do not jar the files into a `META-INF/data` directory in the JAR file itself.

Validating Contents – After bootstrapping data, it is a good idea to validate the contents of what you loaded by using the View functionality of the Datasync Web application.

Pulling Definitions from Production

Developers and testers may be interested in bringing definitions that are being modified in a production environment back into their development domains. As the modified files are stored in the database, Portal provides a mechanism for exporting XML from the database back into files.

One approach is to use the browse capability of the Datasync web application to view all XML stored in the database in a web browser. This information can then be cut and pasted into a file.

A better alternative is to use the DataRepositoryQuery Command Line Tool, which allows you to fetch particular files from the database using an FTP-like interface.

The DataRepositoryQuery Command Line Tool supports a basic, FTP-style interface for querying the data repository on a server.

The command line class is `com.bea.p13n.management.data.DataRepositoryQuery`. In order to use it, you must have the following in your CLASSPATH: `p13n_ejb.jar`, `p13n_system.jar`, and `weblogic.jar`.

Run the class with the argument `help` to print basic usage help.

For example:

```
set classpath=c:\bea\weblogic81\p13n\lib\p13n_system.jar;
c:\bea\weblogic81\p13n\lib\p13n_ejb.jar;
C:\bea\weblogic81\server\lib\weblogic.jar

java com.bea.p13n.management.data.DataRepositoryQuery help
```

Options for Connecting to the Server

Several optional command arguments are used for connecting to the server. The default values are probably adequate for samples provided by BEA. In real deployments, the options will be necessary.

`-username` *userid*	Username of a privileged user (an administrator)	Default = weblogic
`-password` *password*	Password for the privileged user	Default = weblogic
`-app` *appName@host:port*	Application to manage	Default = @7001
`-url` *url*	URL to `DataRepositoryQuery` servlet	

Only one of `-app` or `-url` may be used, as they represent two alternate ways to describe how to connect to a server.

The URL is the most direct route to the server, but it must point to the `DataRepositoryQuery` servlet in the Datasync Web application. This servlet should have the URI of `DataRepositoryQuery`, but you also need to know the hostname, port, and the `context-root` used to deploy `datasync.war` in your application. So the URL might be something like `http://localhost:7001/datasync/DataRepositoryQuery` if datasync.war was deployed with a `context-root` of `datasync`.

The `-app` option allows you to be a bit less specific. All you need to know is the hostname, port number, and the name of the application. If there is only one `datasync.war` deployed, you do not even need to know the application name. The form of the `-app` description is `appname@host:port`, but you can leave out some pieces if they match the default of a single application on localhost port 7001.

The `-app` option can be slow, as it has to perform many queries to the server, but it will print the URL that it finds, so you can use that with the `-url` option on future invocations.

Examples

Assuming CLASSPATH is set as previously described, and the default username/password of weblogic/weblogic is valid):

Find the application named p13nBase running on localhost port 7001:

```
java com.bea.p13n.management.data.DataRepositoryQuery -app p13nBase
```

Find the application named p13nBase running on snidely port 7501:

```
java com.bea.p13n.management.data.DataRepositoryQuery -app
p13nBase@snidely:7501
```

Find the single application running on localhost port 7101:

```
java com.bea.p13n.management.data.DataRepositoryQuery -app @7101
```

Find the single application running on snidely port 7001:

```
java com.bea.p13n.management.data.DataRepositoryQuery -app @snidely
```

Find the single application running on snidely port 7501:

```
java com.bea.p13n.management.data.DataRepositoryQuery -app @snidely:7501
```

In each of the examples, the first line of output will be something like this:
```
Using url: http://snidely:7001/myApp/datasync/DataRepositoryQuery
```

Usage

The easiest way to use the tool is in shell mode. To use this mode, you just invoke DataRepositoryQuery without any arguments (other than those needed to connect as described previously).

In this mode, the tool will start a command shell (you will see a drq> prompt) where you can interactively type commands, similar to how you would use something like ftp.

Alternatively, you can supply a single command (and its arguments), and DataRepositoryQuery will run that command and exit.

Commands

The HELP command will give you help on the commands you can use. Or use HELP *command* to get help on a specific command.

The available commands are:

HELP	Basic Help
HELP OPTIONS	Help on command line options
HELP command	Help on a specific command
HELP WILDCARDS	Help on wildcards that can be used with URI arguments
LIST [-l] [uri(s)]	List available data items
INFO [-l] [-d]	Print repository info
PRINT uri	Print a data item (the xml)
GET [-f] uri [filename]	Retrieve a data item to a file
MGET [-f] [uri(s)]	Retrieve multiple data items as files. Not specifying a URI retrieves all files.
EXIT or QUIT	Exit the shell (shell only)

Commands are not case-sensitive, but arguments such as filenames and data item URIs may be. More help than what is listed above can be obtained by using HELP *command* for the command you are interested in.

Where multiple URIs are allowed (indicated by uri(s) in the help), you can use simple wildcards, or you can specify multiple URIs. The result of the command includes all matching data items.

Options in square brackets ([]) are optional and have the following meanings:

-l	Output a longer, more detailed listing
-d	Include URIs of data items contained in each repository
-f	Force overwrite of existing files

The following example retrieves all assets from the repository as files:

```
java com.bea.p13n.management.data.DataRepositoryQuery -app mget
```

Working with Uncompressed

When working with a production server with an uncompressed EAR, the only difference from development mode is that there is no poller thread.

When updating definition files using the WebLogic Administration Portal, the files are updated on the administration server in the deployed uncompressed EAR directory automatically. This means that the WebLogic Administration Portal can be used from any managed server in the cluster, but the primary store always resides on the administration server. If the deployable EAR directory is read-only, the WebLogic Administration Portal cannot be used to modify files.

Making sure you are not overwriting files – When working with an uncompressed EAR file in production, special care needs to be taken when working with definition files. When you redeploy your application to your production environment, the existing definition files are replaced. If you have administrators updating definitions using the WebLogic Administration Portal, their changes will be lost upon redeploying an updated application.

Copying back to development – To prevent overwriting any changes done by administrators to definition files when redeploying a new portal application, you must first copy all the definition files from the administration server back to development manually or using the Datasync Web application.

Rules for Deploying Datasync Definitions

There are a number of general concepts to think about when iteratively deploying datasync definitions into a production system. In general, adding new datasync definitions to a production system is a routine process that you can do at any time. However, removing or making destructive modifications to datasync definitions can have unintended consequences if you are not careful.

When removing or making destructive modifications to datasync definitions, you should first consider whether there are other components that are linked to those components. There are several types of bindings that might exist between definitions. *For some of these bindings, it is very important to understand that they may have been defined on the production server using the WebLogic Administration Portal and may not be known by the developers*.

One example of bindings is that you may have two datasync definitions bound together. An example of this is a campaign that is based on a user property defined in a user property set. If you remove the property set or the specify property, that campaign will no longer execute properly. In this case, you should update any associated datasync definitions before removing the property set or property.

A second scenario is that you have defined an entitlement rule that is bound to a datasync definition. For example, you might have locked down a portlet based on a dynamic role that

checks if a user has a particular user property value. In this case, you should update that dynamic role before removing the property set or property.

A third scenario is that there are in-page bindings between datasync items and Portal JSP tags. An example is a <pz:contentSelector> tag that references a content selector. Update the content selector tag in the production environment before you remove the content selector. This is one type of binding that is only configured in WebLogic Workshop at development time rather than in the WebLogic Administration Portal.

A good guideline for developers is to not remove or make significant changes to existing datasync definitions that are in production. Instead, create new definitions with the changes that are needed. This can be accomplished by creating new versions of, for example, campaigns where there is no chance that they are being used in unanticipated ways. Additionally, do datasync bootstraps of the production system's existing datasync definitions back into development on a regular basis.

Removing Property Sets

When you remove a property set, any existing values being stored locally by portal in the database will NOT be removed automatically. You need to examine the PROPERTY_KEY and PROPERTY_VALUE tables to clean up the data if desired.

Propagating LDAP and Portal Database Data

The previous sections provided instructions for deploying file-based portal enterprise applications and updating datasync definitions. The WebLogic Portal Propagation Utility lets you propagate application LDAP and database data from one server to another. The Propagation Utility let's you propagate the following data:

Portal database data (DBMS)	LDAP data
Any portal database data created or modified with the WebLogic Administration Portal	• Global roles • Enterprise application roles • Web application roles (delegated administration and visitor entitlement roles) • Delegated administration assignments/definitions • Visitor entitlement assignments/definitions **Note:** The propagation utility lets you view, but not propagate users and groups, because the hashed passwords cannot be propagated.

The propagation utility is a portal Web application packaged inside an enterprise application archive (.ear). The propagation utility is deployed on both the domain containing the LDAP and database data (the source server) and the domain that will receive that data (destination server).

Figure 8 shows the propagation utility interface.

Figure 8   The WebLogic Portal Propagation Utility
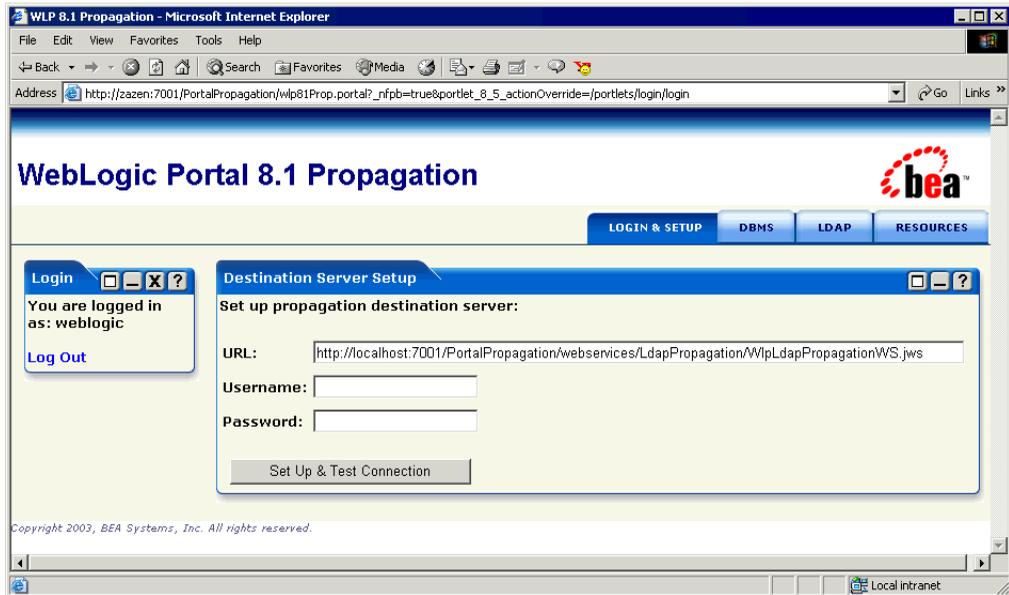
The propagation utility is a self-contained application that does not require you to open or configure it using WebLogic Workshop.

The main use case for the propagation utility is moving data from a staging environment to a production environment. However, another valid use case is moving data from production back to staging in order to simulate the current production environment on staging.

In a clustered environment, propagate only from the source administration server to the destination administration server. The data is then automatically propagated from the destination administration server to the managed servers in the cluster.

Database Requirements – You must propagate between the same type of database, and you must be able to simultaneously connect to the source and destination databases. You must also have a non-transactional (non-XA) database driver for your database installed on the source server.

The following sections show you how to download, deploy, and use the propagation utility.

Getting the Latest Portal Propagation Utility

Contact BEA Support for the latest version of the Propagation Utility.

About the Propagation Utility Files

The propagation utility download contains the following files:

- **readme.html** – Provides abbreviated overview and setup instructions, as well as known issues related to the propagation utility.

- **PortalPropagationEntApp.ear** – The enterprise application archive containing the propagation utility Web application, which must be deployed on the source and destination servers.

- **pdef.zip** – The archive containing the following files to enable portal database propagation (not LDAP propagation):

  - **pdef_81.xml** – Contains the configuration for propagating portal database data. You can modify this file to customize what is propagated. See the help on the Propagation Utility's DBMS portlet for details.

  - **portal-propagation_2_0.xsd** – The scheme for pdef_81.xml. Do not modify this file.

Setting Up and Deploying the Propagation Utility

This section shows you how to set up and deploy the propagation utility. Table 2 provides an overview of the configuration necessary on both the source and destination servers. Detailed instructions follow the table.

Table 2 Propagation Portal configuration on source and destination servers

Source Server	Destination Server
• Modify `/pdef/pdef_81.xml`: enter unique values for the `source-data-source-name` and `destination-data-source-name` attributes to be used for the JDBC data sources. (This is not required if you are propagating only LDAP data.)	• Modify `/pdef/pdef_81.xml`: enter unique values for the `source-data-source-name` and `destination-data-source-name` attributes to be used for the JDBC data sources. (This is not required if you are propagating only LDAP data.)

Table 2 Propagation Portal configuration on source and destination servers

Source Server	Destination Server
• Set up non-XA JDBC connection pools and data sources for the source and destination servers. (This is not required if you are propagating only LDAP data.) This requires a non-XA database driver on the source server.	• No JDBC connection pools or data sources are required, unless you want to propagate from the destination server (such as the production server) to another server (such as a staging server), which would make this a source server as well.
• Set up JMS queues for the propagation utility. The propagation utility does not use JMS.This step lets you prevent a harmless server exception.	• Set up JMS queues for the propagation utility. The propagation utility does not use JMS.This step lets you prevent a harmless server exception.
• Deploy PortalPropagationEntApp.ear.	• Deploy PortalPropagationEntApp.ear.

In the following installation steps, starting with step 6., the order for setting up JDBC, JMS, and deploying the propagation application is arbitrary. You can perform those steps in any order.

To install the WebLogic Portal Propagation Utility:

1. Make sure you have the latest version of the propagation utility from BEA Support.

2. On both the source and destination servers, create a directory for the utility from which you want to deploy the utility's enterprise application, and extract the utility to that directory.

3. After you extract the utility, you will see an archive called `pdef.zip`. Extract this archive to the domain root directory on both the source and destination servers.

   For example, if your destination domain directory on both servers is `/myDomain`, extract `pdef.zip` into that directory on both servers. The `/myDomain/pdef/` directory is created automatically.

4. Open both the source and destination `/pdef/pdef_81.xml` files in an editor.

   In the top-level `<portal-propagation>` element of both files, change the default values of the `source-data-source-name` and `destination-data-source-name` attributes. Make a note of the values you enter. You will use them in later steps to configure the JDBC database connections on the source server.

For example:

```
<portal-propagation
xmlns="http://www.bea.com/portal/xsd/propagation/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/portal/xsd/propagation/2.0
portal-propagation_2_0.xsd"
source-data-source-name="srcDataSource"
destination-data-source-name="destDataSource"
insert-new-records="true"
update-existing-records="true">
```

Save and close the files.

5. Make sure the source and destination servers are running. If you are using clusters for your source and destination, make sure the administration servers are running.

6. Add non-transactional (non-XA) JDBC connections and data sources on the source server. In this step, you will set up a source JDBC connection and data source and a destination JDBC connection and data source, both on the source server. This step requires that you have a non-XA database driver installed for your database.

   Note: The engine that propagates portal database data needs a non-XA connection pool and data source on the source server for both the source and destination databases. Do not attempt to reuse the existing WebLogic Portal connection pool and data source for propagation, because the server will use a transaction manager and attempt to set up the database propagation as a distributed transaction. Propagation will then fail because the propagation engine needs to manage the transaction itself.

   a. Start the WebLogic Server Administration Console on the source server. For example, http://stagingAdminServer:7001/console.

   b. Create the connection pool for the source server. In the left navigation pane, expand **Services > JDBC** and click **Connection Pools**.

   c. In the right pane, click **Configure a new JDBC Connection Pool**.

   d. On the Configure a JDBC Connection Pool page, select your database type, select a non-XA database driver for it, and click **Continue**.

   e. On the page that appears, enter the following:

Name	Any name for the source connection pool.
Database Name	(PointBase) Enter the database name. The default is `workshop`.
Host Name	Enter the source database server name.
Port	Enter the port number of the source database.
Database User Name	Enter the database username for authentication.
Password/Confirm	Enter/confirm the user's password.

f. Click **Continue**. You can then test the database connection by clicking **Test Driver Configuration**.

g. Configure the destination server connection pool on the source server. Click **Connection Pools** in the left pane, click **Configure a new JDBC Connection Pool** in the right pane, select your database type and the non-XA driver, and click **Continue**.

h. On the page that appears, enter the following:

Name	Any name for the destination connection pool.
Database Name	(PointBase) Enter the database name. The default is `workshop`.
Host Name	Enter the destination database server name.
Port	Enter the database port on the destination server.
Database User Name	Enter the database username for authentication.
Password/Confirm	Enter/confirm the user's password.

i. Click **Continue**. You can then test the database connection by clicking **Test Driver Configuration**.

Your connection pools are configured. Now configure your data sources on the source server to correspond to the connection pools.

j. Create the data source for the source database. In the left navigation pane, expand **Services > JDBC** and click **Data Sources**.

k. In the right pane, click **Configure a new JDBC Data Source**.

l. On the Configure a JDBC Data Source page, in both the **Name** and **JNDI Name** fields, enter the value of the `source-data-source-name` attribute you entered in the `/pdef/pdef_81.xml` file. For example, enter `srcDataSource`.

m. Deselect the **Honor Global Transactions** option to make the data source non-XA, as shown in Figure 9.

Figure 9   Deselect Honor Global Transactions to make the data source non-XA
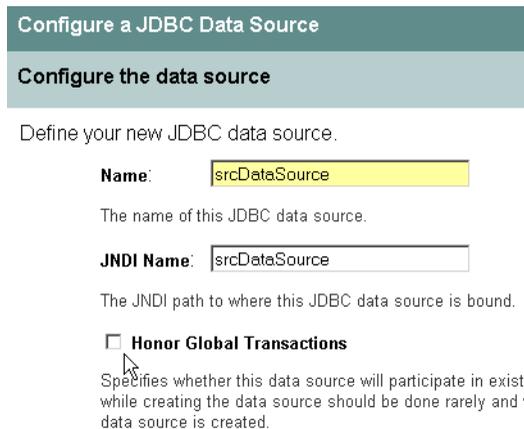
n. Click **Continue**.

o. On the page that appears, select the corresponding source connection pool (for example, srcPool), and click **Continue**.

p. On the page that appears, select the target server (the administration server in a cluster), and click **Create**.

q. Now create a second data source on the source server for the destination database. In the left navigation pane, expand **Services > JDBC** and click **Data Sources**, and in the right pane click **Configure a new JDBC Data Source**.

r. On the Configure a JDBC Data Source page, in both the **Name** and **JNDI Name** fields, enter the value of the `destination-data-source-name` attribute you entered in the `/pdef/pdef_81.xml` file. For example, enter `destDataSource`.

s. Deselect the **Honor Global Transactions** option to make the data source non-XA, as shown in Figure 9.

t. Click **Continue**.

u. On the page that appears, select the corresponding source connection pool (for example, destPool), and click **Continue**.

v. On the page that appears, select the target server (the administration server in a cluster), and click **Create**.

Connection pool and data source setup is complete. Create connection pools and data sources on any servers that will serve as source servers. For example, set up connection pools and data sources on the production server if you want to propagate back to a staging server.

7. Create JMS queues for the propagation utility on both servers. While the propagation utility does not use JMS, performing these steps eliminates JMS exceptions that do not affect propagation.

   a. Start the WebLogic Server Administration Console on one of the servers. For example, http://productionAdminServer:7001.

   b. In the left navigation pane, expand **Services > JMS > Servers > cgJMSServer** (or the server name you used when you set up the domain), and click **Destinations**.

   > **Note:** You may see a list of existing JMS queues you have configured. You must still create the JMS queues for the propagation utility because it runs in its own Web application.

   c. In the right pane, click **Configure a new JMS Queue**.

   d. In Create a New JMS Queue window, on the General tab, enter the following in the **Name** and **JNDI Name** fields:

   ```
   PortalPropagation.queue.AsyncDispatcher_error
   ```

   e. Click **Create**.

   f. Click the **Redelivery** tab and change the **Redelivery Limit** value to 0 (zero).

   g. Click **Apply**.

   h. Create a second JMS Queue. Click **Destinations** in the left pane, and click **Configure a new JMS Queue** in the right pane.

   i. Enter the following in the **Name** and **JNDI Name** fields:

```
PortalPropagation.queue.AsyncDispatcher
```

j. Click **Create**.

k. Click the **Redelivery** tab, and in the **Error Destination** field select
`PortalPropagation.queue.AsyncDispatcher_error`.

l. Click **Apply**. The JMS queues on the first server are configured.

m. On the other server (source or destination), perform the previous steps to set up the two JMS queues.

8. Deploy the propagation utility `PortalPropagationEntApp.ear` on the source and destination servers.

a. Start the WebLogic Server Administration Console on the source server. For example, http://stagingAdminServer:7001/console.

b. In the left navigation pane, expand **Deployments** and click **Applications**.

c. In the right pane, click **Deploy a new Application**.

d. In the Deploy a New Application window, use the Location links to select the directory containing `PortalPropagationEntApp.ear` (the directory to which you extracted the propagation utility).

e. When `PortalPropagationEntApp.ear` appears in the list, select the option button next to it and click **Continue**.

f. On the page that appears, click **Deploy**.

g. On the page that appears, deployment is complete when the Status of Last Action column shows the message "Success."

h. On the other server (source or destination), perform the previous steps to deploy `PortalPropagationEntApp.ear`.

Note: **Windows only** – If you receive the following type of exception while trying to deploy:

```
java.lang.InternalError: IO error while trying to compute name
from: <path>
```

it probably means the path to your server exceeds 254 characters. As a workaround, shorten the path to the server for the domain receiving the exception. For example, if your domain path is `C:\bea\user_projects\domains\destinationDomain`, move the domain to something like `C:\destinationDomain`.

Starting and Using the Propagation Utility

1. Start the propagation utility on the source server. Enter the following URL in a browser: http://<host>:<port>/PortalPropagation/wlp81Prop.portal. (The propagation utility does not have to be started on the destination server. It must only be deployed.)

2. On the Login & Setup page, log in as a user in the Administrators user group.

3. In the Destination Server Setup portlet, change `localhost:7001` to the host:port of the destination domain, and enter the username and password of a member of the Administrators group in the destination domain. **Click Set Up & Test Connection** to connect to the destination server.

4. Go to the DBMS page to propagate portal database data to the destination server.

5. Go to the LDAP page to propagate roles, delegated administration definitions, and visitor entitlement definitions, or view users and groups (read-only).

Help with the Portlets – The propagation portlets are self-documented. For information on using a portlet, click the help icon on the portlet titlebar.

Understanding Portal Resources

The Portal Library contains books, pages, layouts, portlets, desktops, and other types of portal-specific resources. Using the WebLogic Administration Portal, these resources can be created, modified, entitled, and arranged to shape the portal desktops that end users access.

Figure 10 shows an image of the portal resource tree in the WebLogic Administration Portal. The library contains the global set of portlets and other resources, while the Portals node contains instances of those resources, such as the colorsSample desktop and its pages, books, and portlets.

Figure 10 Portal resources library
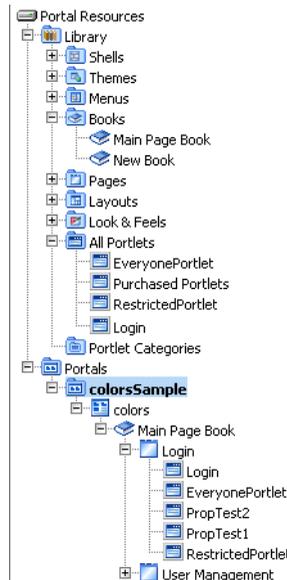
Each of these resources is defined partially in the portal database so they can be easily modified at run time. The majority of resources that exist are created by an administrator, either from scratch or by creating a new desktop from an existing `.portal` template file that was created in WebLogic Workshop.

However, portlets themselves are created by developers and exist initially as XML files. In production, any existing `.portlet` files in a portal application are automatically read into the database so they are available to the WebLogic Administration Portal.

The following section addresses the lifecycle and storage mechanisms around portlets, since their deployment process is an important part of portal administration and management.

Portlet Deployment Lifecycle

During development time, `.portlet` files are stored as XML in any existing portal Web application in the Portal EAR. As a developer creates new `.portlet` files, a file poller thread monitors changes and loads the development database with the `.portlet` information.

In a production environment, `.portlet` files are loaded when the portal Web application that contains them is redeployed on the administration server. This redeployment timing ensures that

the content of the portlet, such as a JSP or Page Flow, is available at the same time as the `.portlet` file is available in the Portal Library. The administration server is the chosen master responsible for updating the database so that there are no contention issues around every server in the production cluster trying to write the new portlet information into the database at the same time. When deploying new portlets to a production environment, target the portal application for redeployment on the administration server.

Database Structure for Storing Portlets

When a portlet is loaded into the database, the portlet XML is parsed and a number of tables are populated with information about the portlet, including PF_PORTLET_DEFINITION, PF_MARKUP_DEFINITION, PF_PORTLET_INSTANCE, PF_PORTLET_PREFERENCE, L10N_RESOURCE, and L10N_INTERSECTION.

PF_PORTLET_DEFINITION is the master record for the portlet and contains rows for properties that are defined for the portlet, such as the definition label, the forkable setting, edit URI, help URI, and so on. The definition label and Web application name are the unique identifying records for the portlet. Portlet definitions refer to the rest of the actual XML for the portlet that is stored in PF_MARKUP_DEF.

PF_MARKUP_DEF contains stored tokenized XML for the `.portlet` file. This means that the `.portlet` XML is parsed into the database and properties are replaced with tokens. For example, here is a snippet of a tokenized portlet:

```
<netuix:portlet $(definitionLabel) $(title) $(renderCacheable)
$(cacheExpires)>
```

These tokens are replaced by values from the master definition table in PF_PORTLET_DEFINITION, or by a customized instance of the portlet stored in PF_PORTLET_INSTANCE.

The following four types of portlet instances are recorded in the database for storing portlet properties:

- **Primary** – Properties defined in development and stored in the .portlet file.

- **Library** – Properties defined in the Portal Library, which may be changed using the WebLogic Administration Portal.

- **Admin** – A customized instance of the portlet in a desktop. This allows you to customize a portlet in a particular way for a desktop without affecting other instances of the portlet in other desktops.

- **User** – User-customized instances of the portlet defined in the Visitor Tools.

PF_PORTET_INSTANCE contains properties for the portlet for attributes such as DEFAULT_MINIMIZED, TITLE_BAR_ORIENTATION, and PORTLET_LABEL.

If a portlet has portlet preferences defined, those are stored in the PF_PORTLET_PREFERENCE table.

Finally, portlet titles can be internationalized. Those names are stored in the L10N_ RESOURCE table which is linked using L10N_INTERSECTION to PF_PORTLET_DEFINITION.

Removing Portlets from Production

If a portlet is removed from a newly deployed portal application, and it has already been defined in the production database, it is marked as IS_PORTLET_FILE_DELETED in the PF_PORTLET_DEFINITION table. It will then show up as grayed out in the WebLogic Administration Portal, and user requests for the portlet if it is still contained in a desktop instance will return a message that says the portlet is unavailable.

Zero Downtime Architectures

One limitation of redeploying a portal application to a WebLogic cluster is that during redeployment users cannot access the site. For enterprise environments where it is not possible to schedule down time to update a portal application with new portlets and other components, a multi-cluster configuration lets you keep your portal application up and running during redeployment.

The basis for a multi-clustered environment is the notion that you have a secondary cluster to which user requests are routed while you update the portal application in your primary cluster.

For normal operations, all traffic is sent to the primary cluster, as shown in Figure 11. Traffic is not sent to the secondary cluster under normal conditions because the two clusters cannot use the same session cache. If traffic was being sent to both clusters and one cluster failed, a user in the middle of a session on the failed cluster would be routed to the other cluster, and the user's session cache would be lost.

Figure 11   During normal operations, traffic is sent to the primary cluster
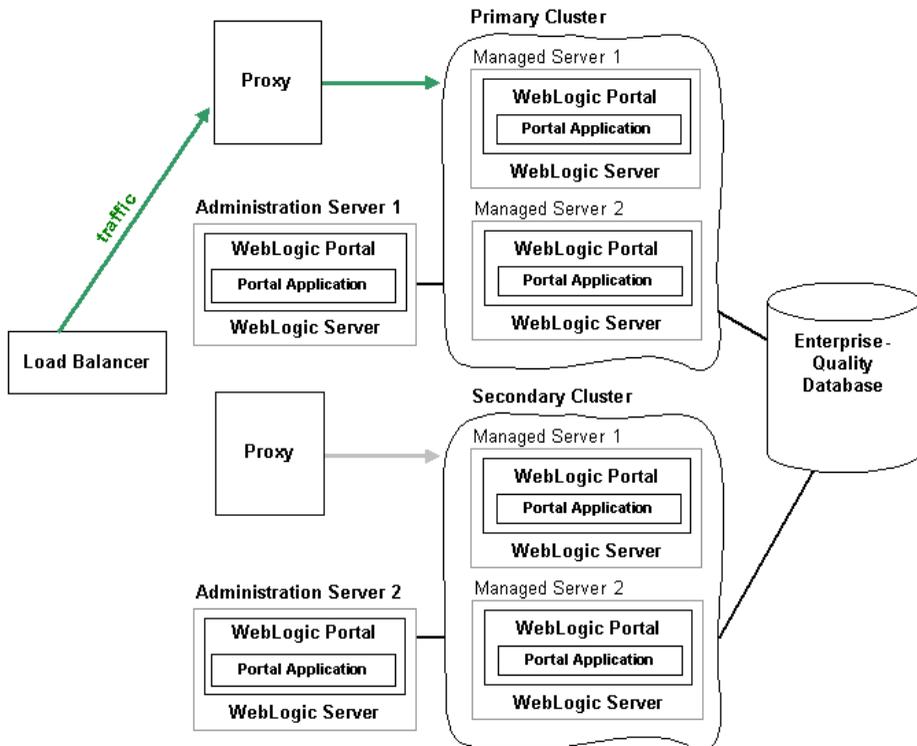
Step 1 – All traffic is routed to the secondary cluster, then the primary cluster is updated with a new Portal EAR, as shown in Figure 12. This EAR has a new portlet, which is loaded into the database.

Figure 12   Traffic is routed to the secondary cluster; the primary cluster is updated
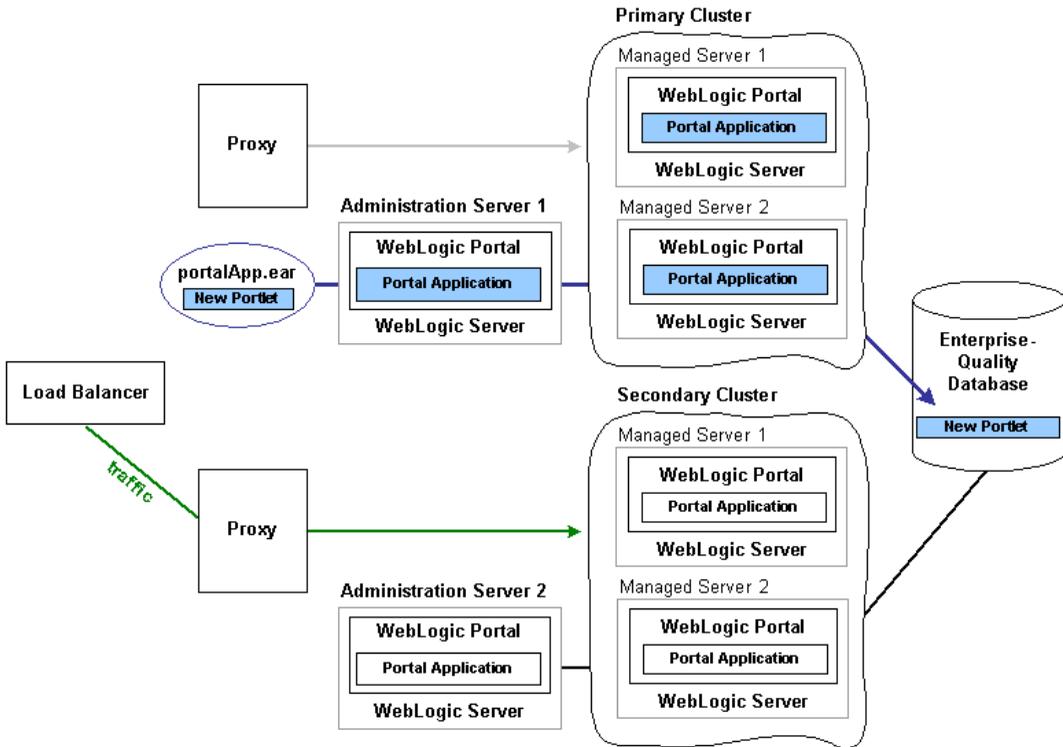
Routing requests to the secondary cluster is a gradual process. Existing requests to the primary cluster must first end over a period of time until no more requests exist. At that point, you can update the primary cluster with the new portal application.

Step 2 – All traffic is routed back to the primary cluster, and the secondary cluster is updated with the new EAR, as shown in Figure 13. Because the database was updated when the primary cluster was updated, the database is not updated when the secondary cluster is updated.

Figure 13 Traffic is routed back to the primary cluster; the secondary cluster is updated
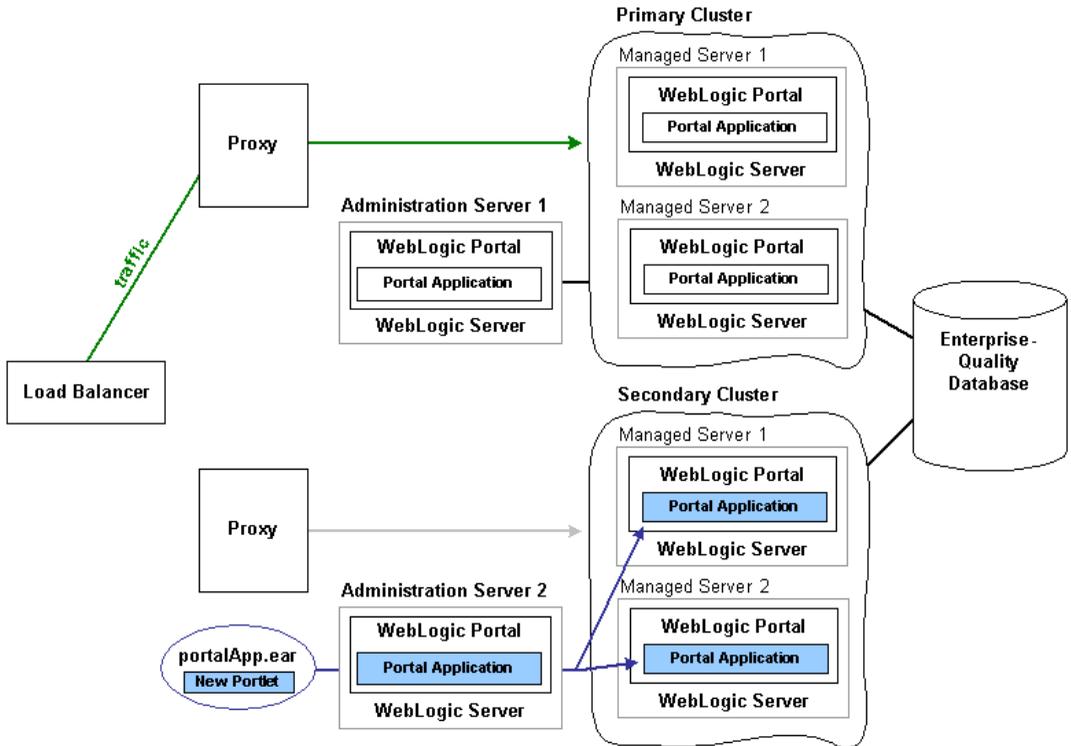
Even though the secondary cluster does not receive traffic under normal conditions, you must still update it with the current portal application. When you next update the portal application, the secondary cluster will temporarily receive requests, and the current application must be available.

In summary, to upgrade a multi-clustered portal environment, you switch traffic away from your primary cluster to a secondary one that is pointed at the same portal database instance.   You can then update the primary cluster and switch users back from the secondary. This switch can happen instantaneously, so the site experiences no down time. However, in this situation, any existing user sessions will be lost during the switches.

A more advanced scenario is a gradual switchover, where you switch new sessions to the secondary cluster, and after the primary cluster has no existing user sessions you upgrade it. Gradual switchovers can be managed using a variety of specialized hardware and software load balancers. For both scenarios, there are several general concepts that should be understood before

deploying applications, including the portal cache and the impact of using a single database instance.

Single Database Instance

When you configure multiple clusters for your portal application, they will share the same database instance. This database instance stores configuration data for the portal. This can become an issue, because when you upgrade the primary cluster it is common to make changes to portal configuration information in the database. These changes are then picked up by the secondary cluster where users are working.

For example, redeploying a portal application with a new portlet to the primary cluster will add that portlet configuration information to the database. This new portlet will in turn be picked up on the secondary cluster. However, the new content (JSP pages or Page Flows) that is referenced by the portlet is not deployed on the secondary cluster.

Portlets are only invoked when they are part of a desktop, so having them available to the secondary cluster will have no immediate effect on the portal that users see. However, adding a new portlet to a desktop with the WebLogic Administration Portal will immediately affect the desktop that users see on the secondary cluster. In this case, that portlet would show up, but the contents of the portlet will not be found.

To handle this situation you have several options. First, you can delay adding the portlet to any desktop instances until all users are back on the primary cluster. Another option is to entitle the portlet in the library so that it will not be viewable by any users on the secondary cluster. Then add the portlet to the desktop, and once all users have been moved back to the primary cluster, remove or modify that entitlement.

A special case to be aware of is if you are updating an existing portlet's content URI to a new location that is not yet deployed. For this reason, updating the content URI of a portlet should be done with care or as part of a multi-phase update.

Another important consideration when running two portal clusters simultaneously against the same database is the portal cache.

Portal Cache

WebLogic Portal provides facilities for a sophisticated cluster-aware cache. This cache is used by a number of different portal frameworks to cache everything from markup definitions to portlet preferences. Additionally, developers can define their own caches using the portal cache framework. The portal cache is configured in the WebLogic Administration under Configuration

Settings / Service Administration / Cache Manager. For any cache entry, the cache can be enabled or disabled, a time to live can be set, the cache maximum size can be set, the entire cache can be flushed, or you can invalidate a specific key.

When a portal framework asset that is cached is updated, it will typically write something to the database and automatically invalidate the cache across all machines in the cluster. This process keeps the cache in sync for users on any managed server.

When operating a multi-clustered environment for application redeployment, special care needs to be taken with regard to the cache. The cache invalidation mechanism does not span both clusters, so it is possible to make changes on one cluster that will be written to the database but not picked up immediately on the other cluster. As this situation could lead to system instability, it is recommended that during this user migration window the caches be disabled on both clusters. This is important when you have a gradual switchover between clusters versus a hard switch that drops existing user sessions.