



The banner features a red background. On the left, there is a photograph of a modern glass skyscraper. On the right, a dark gray square contains a silver film reel icon and the text "Video Placeholder Your video will display here." Below the images, the Oracle logo is displayed in red, followed by "WebLogic Server 12c" in black. The main title "Oracle WebLogic Server 12c: Deploy Java EE Applications" is in red, with the presenter's name "Vijaya Karothi, Curriculum Developer" in black below it. A circular "ORACLE THIRTY YEARS 30" anniversary logo is in the top right. A red bar at the bottom contains the Oracle logo and a copyright notice.

ORACLE®

WebLogic Server 12c

Oracle WebLogic Server 12c: Deploy Java EE Applications

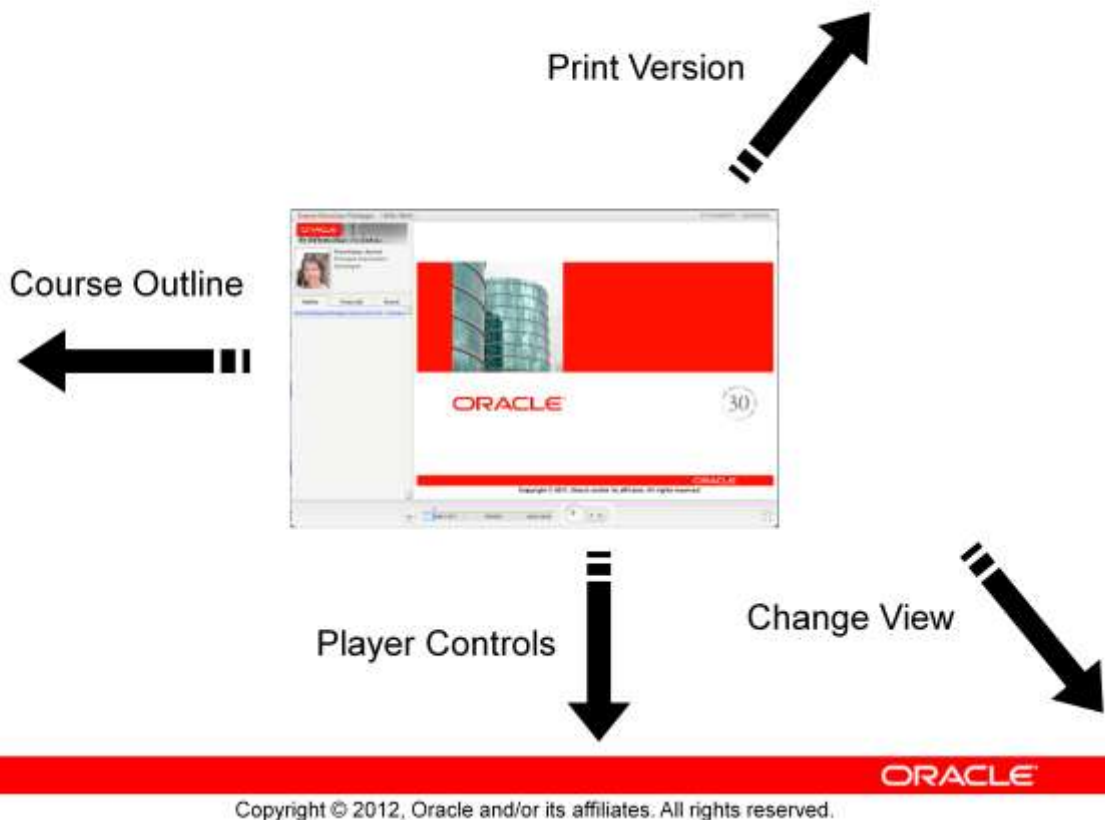
Vijaya Karothi, Curriculum Developer

ORACLE®

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Hello, and welcome to this online, self-paced course entitled Oracle WebLogic Server 12c: Deploy Java EE Applications. My name is TJ Palazzolo and I will be your guide for approximately the next 90 minutes of interactive lectures and review sessions. The aim of this course is to get you familiar with the application packaging, deployment and management features of WebLogic Server. Note, however, that this course is not intended to address the development of Java EE applications.

Using the Player



Before we begin, take a look at some of the features of this Flash-based course player. If you've attended a similar self-paced course in the past, then feel free to skip this slide.

Let's start with the Outline. It's set up so that you can go at your own pace. Feel free to skip over topics you already feel confident about, or jump right to a feature that really interests you, or go back and review topics that were already covered. Just click a slide title in the outline to display its contents. By default we will automatically walk you through the slides without requiring you to use the outline. Use the Transcript tab to view the audio transcript for each slide, and use the Search tab to find specific information in the course.

Next, let's look at the Player Controls. Use these controls to Pause, Play, or move to the Previous or Next slides. You can also use the interactive Progress Bar to fast forward or rewind the current slide. Some interactive slides in this course may contain additional navigation and controls.

This icon on the bottom right corner allows you to change the View of this player. Some parts of the course are programmed to open in full-screen mode, but you can control the view at any time using this button.

A downloadable, Print Version of this course is available from the Attachments button.

This course will now pause, so feel free to take some time and explore the interface. Then when you're ready to continue, click the Next button below or alternatively click the About This Course slide in the outline on the left.



So, you know the title of the course, but you may be asking yourself, "Am I in the right place?" To help you answer this question, you can access information here regarding the course objectives, the target audience, and the prerequisites. When finished, click the Next Slide button.

Road Map



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Now let's get started. The first lesson in this self-study reviews basic Java EE application concepts, packaging, and configuration, and also introduces you to some related concepts specific to WebLogic Server.

Objectives

After completing this lesson, you should be able to:

- Describe the components of a typical Java EE system
- List several types of Java EE applications
- Describe the Java EE packaging structure
- Explain some capabilities found in WebLogic deployment descriptors
- Discuss the benefits of WebLogic shared libraries

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

To be more specific, we'll talk about the components of a typical Java EE system, such as Web applications, EJB applications, and Web services. For each application type, we'll review the basic packaging structure and also identify some optional WebLogic configuration files, known as deployment descriptors. This includes a brief explanation of these descriptors' capabilities. Lastly, this lesson introduces the shared library feature of WebLogic server.

A Web Application:

- Responds to client requests using the HTTP protocol
- Typically implements an interactive Web site

The contents of a Web application can include:

- Java Servlets
- JavaServer Pages (JSPs) for dynamic content
- Static content (HTML, CSS, images, and so on)
- Java classes and libraries
- Client-side libraries (JavaScript, Java Applets, and so on)
- XML deployment descriptors

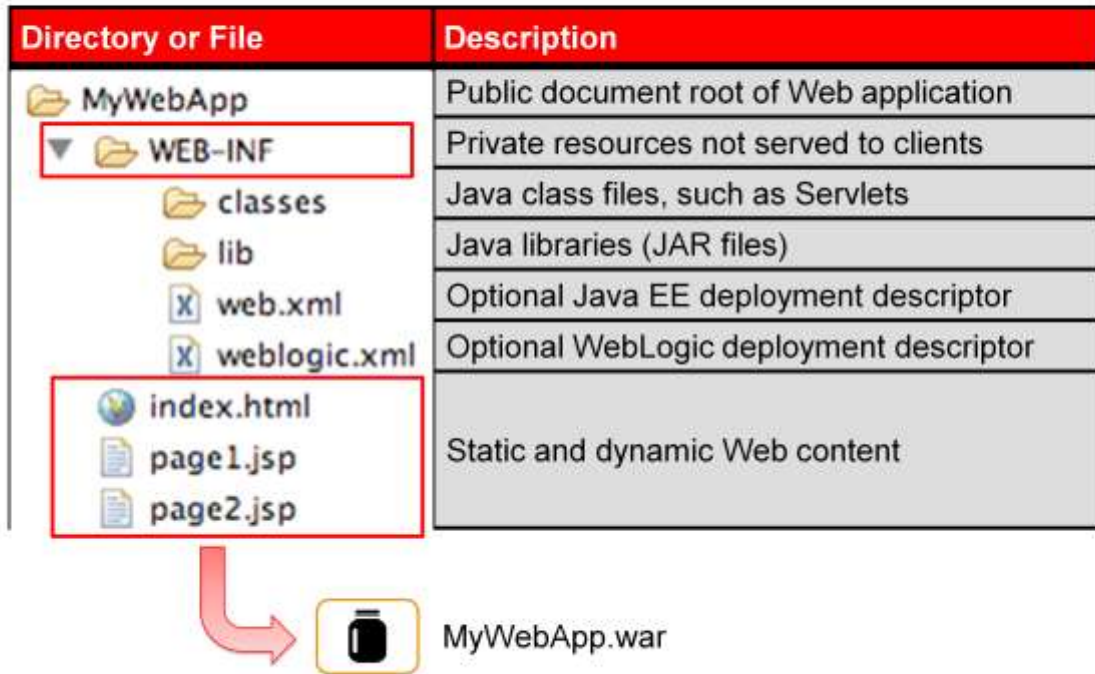
A Web application on WebLogic Server includes the following files:

- Servlets, JSPs, and other Java classes
- Optionally, a `web.xml` deployment descriptor, which is a Java EE standard XML document that describes the contents of a WAR file
- Optionally, a `weblogic.xml` deployment descriptor, which is an XML document containing WebLogic Server-specific elements for Web applications
- A Web application can also include HTML and XML pages with supporting files, such as images and multimedia files

A servlet is a Java class that runs in a Java-enabled server, handles an HTTP request and provides an HTTP response, usually in the form of an HTML page. The most common use of HTTP Servlets is to create interactive applications using standard Web browsers for the client-side presentation. HTTP servlets can access databases, Enterprise JavaBeans, messaging APIs, HTTP sessions, and other facilities of WebLogic Server.

JavaServer Pages (JSPs) are Web pages coded with an extended HTML that makes it possible to embed Java code on a Web page. JSPs can call custom Java classes, known as tag libraries, using HTML-like tags. The appc compiler compiles JSPs and translates them into servlets. WebLogic Server automatically compiles JSPs if the servlet class file is not present or is older than the JSP source file. You can also precompile JSPs and package the servlet class in a Web Application (WAR) file to avoid compiling in the server.

Web Application Structure



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

You develop your Web application within a specified directory structure so that it can be archived and deployed on WebLogic Server, or another Java EE compliant server. All servlets, classes, static files, and other resources belonging to a Web application are organized under a directory hierarchy. The root of this hierarchy defines the document root of your Web application. All files under this root directory can be served to the client, except for files under `WEB-INF`. All files under `WEB-INF` are private and are not served to a client, including XML deployment descriptors.

A Web archive (WAR file) contains the files that make up a Web application. A WAR file is deployed as a unit on one or more WebLogic Server instances. The WAR file can be deployed alone or packaged in an Enterprise application archive (EAR file) with other application components. If deployed alone, the archive must end with a `.war` extension. If deployed in an EAR file, the archive must end with an `.ear` extension. Alternatively, WebLogic allows you to deploy the Web application directory without archiving it. This technique is especially useful while the application is under development.

Using `weblogic.xml`, you can configure:

- The application's root context path
- Application logging
- Security role mappings
- Advanced session settings
- Session clustering
- References to shared libraries
- References to server resources (data sources, EJBs, etc.)
- Work managers and threading
- Virtual directories
- JSP compiler options

The `weblogic.xml` deployment descriptor follows a proprietary schema used only by WebLogic Server. It allows you to enable and configure Web application features that are not part of the Java EE specification. For example:

- Change the default root URL path of the Web application.
- Direct application log messages to a dedicated log file.
- Change the default HTTP session timeout.
- Change the default cookie name used to track HTTP sessions.
- Enable clustering features, such as in-memory replication and persistence.
- Assign a WebLogic work manager to process requests to this application.
- Tune the threading behavior used to process requests to this application.
- Map other file system locations to URLs for this Web application.
- Enable JSP precompilation.
- Enable directory index pages.
- Enable dynamic reloading of classes.
- Set the default MIME type.

A Web Service Application:

- Responds to HTTP client requests using the Simple Object Access Protocol (SOAP)
- Uses the same structure as a Java EE Web application
- Supports two additional deployment descriptors:
 - `webservices.xml`
 - `weblogic-webservices.xml`

Web Services can be shared by and used as modules of distributed Web-based applications. They commonly interface with existing back-end applications, such as customer relationship management systems, order-processing systems, and so on. Web Services can reside on different computers and can be implemented by vastly different technologies, but they are packaged and transported using standard Web protocols, such as HTTP, thus making them easily accessible by any user on the Web. WebLogic Web services are typically packaged as Java EE Web applications.

The Java EE Web service programming model allows you to create an annotated Java file and then use Ant tasks to compile the file into a Java class and generate all the associated artifacts. The Java Web Service (JWS) annotated file is the core of your Web Service. It contains the Java code that determines how your Web Service behaves. The JWS annotations you can use in a JWS file include the standard ones defined by the Web Services Metadata for the Java Platform specification as well as a set of other standard or WebLogic-specific annotations, depending on the type of Web Service you are creating.

An application that configures one or more Web Service endpoints is called a Web Service Application. The standard Java EE deployment descriptor for Web Services is called `webservices.xml`. This file specifies the set of Web Services that are to be deployed to WebLogic Server and the dependencies they have on container resources and other services.

Enterprise JavaBeans (EJBs):

- Standardize the development and deployment of server-side, distributed components
- Are annotated Java classes
- Are packaged with XML deployment descriptors
- Support the following capabilities:
 - Remote access over a network
 - Object-relational mapping via WLS or the Java Persistence API (JPA)
 - Transactions
 - Messaging integration
 - Dependency injection

Enterprise JavaBeans (EJBs) beans are server-side Java modules that implement a business task or entity and are written according to the EJB specification. There are three types of EJBs: session beans, entity beans, and message-driven beans.

One of the central goals of the EJB specification is to make it easier to program Java components, in particular by reducing the number of required programming artifacts and introducing a set of EJB-specific metadata annotations that make programming the bean file easier and more intuitive. Another goal of the EJB specification is to standardize the persistence framework and reduce the complexity of the entity bean programming model and object-relational (O/R) mapping model.

Java EE cleanly separates the development and deployment roles to ensure that modules are portable between EJB servers that support the EJB specification. Deploying an EJB in WebLogic Server requires running the WebLogic Server appc compiler to generate classes that enforce the EJB security, transaction, and life cycle policies.

EJB Application Structure

| Directory or File | Description |
|----------------------|---|
| MyEJBApp | Application root folder |
| javapackage | Java classes organized into packages |
| MyEJB1.class | EJB and other Java class files |
| MyEJB2.class | |
| META-INF | |
| ejb-jar.xml | Optional Java EE and JPA deployment descriptors |
| persistence.xml | |
| weblogic-cmp-jar.xml | Optional WebLogic deployment descriptors |
| weblogic-ejb-jar.xml | |



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Unlike prior versions of the EJB specification, you are no longer required to create the EJB deployment descriptor files. You can now use metadata annotations in the Java bean file itself to configure metadata. You are still allowed, however, to use XML deployment descriptors if you want; in the case of conflicts, the deployment descriptor value overrides the annotation value.

The Java EE-specified deployment descriptor, `ejb-jar.xml`, describes the enterprise beans packaged in an EJB application. It defines the beans' types, names, and the names of their home and remote interfaces and implementation classes. The `ejb-jar.xml` deployment descriptor defines security roles for the beans, and transactional behaviors for the beans' methods.

Additional deployment descriptors provide WebLogic-specific deployment information. A `weblogic-cmp-rdbms-jar.xml` deployment descriptor, which is unique to container-managed entity beans, maps a bean to tables in a database. The `weblogic-ejb-jar.xml` deployment descriptor supplies additional information specific to the WebLogic Server environment, such as JNDI bind names, clustering, and cache configuration.

EJB modules are packaged as archive files having a `.jar` extension, but can also be deployed as exploded archive directories.

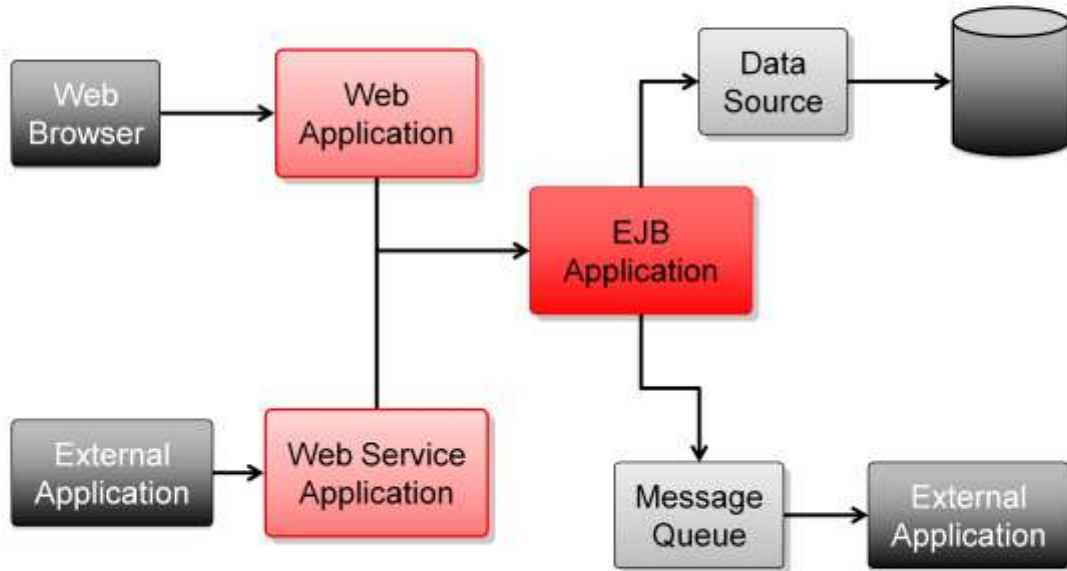
Using `weblogic-ejb-jar.xml`, you can configure:

- Security role mappings
- Advanced security settings
- EJB clustering
- EJB pooling and caching
- Work managers and threading

The `weblogic-ejb-jar.xml` deployment descriptor follows a proprietary schema used only by WebLogic Server. It allows you to enable and configure EJB features that are not part of the Java EE specification. For example:

- Map security role names used in EJB annotations to identities in the WebLogic security realm.
- Run an EJB within the context of a specific WebLogic security identity.
- Enable load balancing and failover for remote EJB invocations.
- Tune EJB performance using pool and cache settings.
- Assign a WebLogic work manager to process requests to an EJB.
- Tune the threading behavior used to process requests to an EJB.

A Typical Java EE System



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

This diagram depicts a typical enterprise system build using Java EE and WebLogic Server:

- An EJB application contains distributed components to model the core business logic. Some EJBs use object relational mappings to retrieve and update business data.
- A JDBC Data Source manages the connections to the back-end database.
- The EJBs use the Java Message Service to publish messages to other external Java EE and non-Java EE applications. EJBs may also subscribe to and process messages published from elsewhere in the enterprise.
- Users interact with the system through an HTML-based Web Application, delivered via a Web browser. This Web application interacts with EJBs to perform business logic.
- To integrate other Java EE and non-Java EE systems within the enterprise, the business logic modeled in the EJBs is also exposed as a collection of Web Services.

Because of the distributed nature of Java EE and WebLogic Server, the components of this enterprise application can be deployed to a single server, to multiple servers, or even across multiple WebLogic domains. For example, the EJB application, Web application, and Web service application could all be distributed onto separate, dedicated server instances.

An Enterprise Application:

- Comprises one or more Java EE application modules:
 - Web Applications
 - EJB Applications
 - Other Java libraries (JARs)
- Allows related applications to be deployed as a unit
- Can include application-specific JDBC and JMS resources

A Java EE Enterprise application consists of one or more Web application modules, EJB modules, Connector modules and other libraries, which can be managed and deployed as a single unit.

JMS and JDBC configurations are stored as modules as well, which are similar to standard Java EE modules. An administrator can create and manage JMS and JDBC modules as global system resources, as modules packaged with a Java EE application (as a packaged resource), or as stand-alone modules that can be made globally available.

For both production and development purposes, Oracle recommends that you package and deploy even stand-alone Web applications, EJBs, and resource adapters as part of an Enterprise application. Doing so allows you to take advantage of Oracle's split development directory structure, which greatly facilitates application development.

Enterprise Application Structure

| Directory or File | Description |
|--------------------------|--|
| MyApp | Application root folder |
| APP-INF | |
| classes | Common Java class files |
| lib | Common Java libraries (JARs) |
| META-INF | |
| application.xml | Optional Java EE deployment descriptor |
| mydatasource-jdbc.xml | JDBC and JMS modules |
| myqueue-jms.xml | |
| weblogic-application.xml | Optional WebLogic descriptor |
| EJBApp.jar | |
| WebApp1.war | |
| WebApp2.war | |
| | Web and EJB application modules |



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Enterprise applications can be deployed as a directory structure or packaged as an archive file with the extension `ear`.

The optional `META-INF/application.xml` deployment descriptor contains an element for each Web application, EJB, and connector module, as well as additional elements to describe security roles and application resources such as databases. If this descriptor is present, the WebLogic deployer picks the list of modules from this descriptor. However, if this descriptor is not present, the container guesses the module names and types from the packaging structure and the annotations defined in Java classes.

By default, Java classes present in EJB modules are accessible by any Web application module found in the same enterprise application. However, Java classes within Web application modules are not available to other Web application modules. To share class files and libraries among all modules, place them within the WebLogic-specific `APP-INF` directory of the enterprise application. These resources will be made available to all application modules within the enterprise application.

Using `weblogic-application.xml`, you can configure:

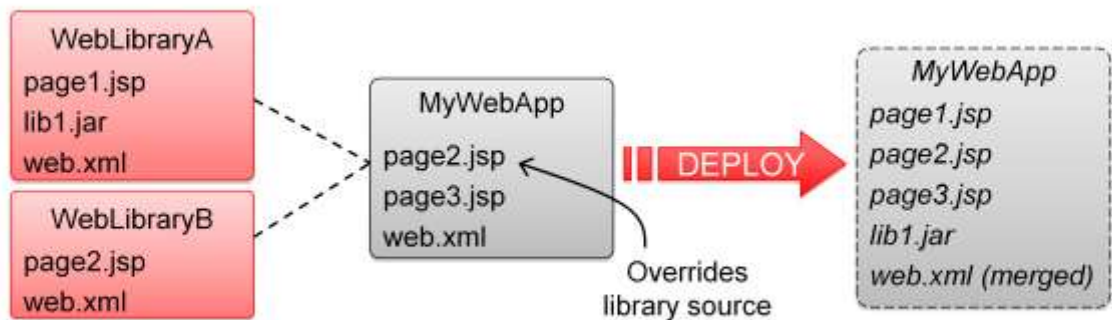
- References to shared libraries
- Work managers and threading
- Default EJB and Web application parameter values

The `weblogic-application.xml` file is the WebLogic Server-specific deployment descriptor extension for the standard Java EE `application.xml` deployment descriptor. It allows you to enable and configure features such as:

- Referencing a shared library module deployed outside of this application
- Assigning a WebLogic work manager to process requests to the application modules
- Tuning the threading behavior used to process requests to the application modules
- Changing the default HTTP session timeout for all Web application modules
- Changing the default cookie name used to track HTTP sessions for all Web application modules
- Enabling clustering features such as in-memory replication and persistence for all Web application modules
- Configuring an applicationwide EJB cache

A Java EE *Shared Library*:

- Is a reusable portion of a Web or Enterprise application
- Is referenced by other deployed applications
- Avoids duplicating source files among Java EE projects
- Can contain deployment descriptors that are merged with the application's descriptors



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

A Java EE Shared Library is a reusable portion of a Java EE enterprise application or Web application. At the enterprise application level, a shared library is an EAR file that can include Java classes, EJB deployments, and Web applications. At the Web application level, a shared library is a WAR file that can include servlets, JSPs, and tag libraries. Shared libraries can be included in an application by reference, and multiple applications can reference a single shared library.

You can deploy as many shared libraries to WebLogic Server as you require. In turn, libraries can reference other libraries, and so on. Because the shared library code and your own application code are assembled at run time, rules must exist to resolve potential conflicts. These rules are the following:

- Any file that is located in your application takes precedence over a file that is in a shared library.
- Conflicts arising between referenced libraries are resolved based on the order in which the libraries are specified in the `META-INF/weblogic-application.xml` file (for enterprise applications) or the `WEB-INF/weblogic.xml` file (for Web applications).

When a deployed enterprise application references one or more shared libraries, the server internally merges the information in the `weblogic-application.xml` file of the referencing enterprise application with the information in the deployment descriptors of the referenced libraries. Similar merging occurs for Web application deployment descriptors.

Shared Library References

- For Web applications, list required shared libraries in `weblogic.xml`.
- For enterprise applications, list required shared libraries in `weblogic-application.xml`.

Excerpts from `weblogic.xml`:

```
<library-ref>
  <library-name>ajax-tools-lib</library-name>
  <specification-version>1.5.0</specification-version>
  <implementation-version>2.0.0</implementation-version>
</library-ref>

<library-ref>
  <library-name>help-web-lib</library-name>
  <specification-version>1.5.0</specification-version>
  <implementation-version>1.1.0</implementation-version>
</library-ref>
```

Shared library
name and version

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

WebLogic Server supports versioning of shared Java EE libraries, so that referencing applications can specify a required minimum version of the library to use, or an exact, required version. WebLogic Server supports two levels of versioning for shared Java EE libraries:

- The Specification Version identifies the version number of the specification (for example, the Java EE specification version) to which a shared Java EE library or optional package conforms.
- The Implementation Version identifies the version number of the actual code implementation for the library or package. For example, this would correspond to the actual revision number or release number of your code. Note that you must also provide a specification version to specify an implementation version.

As a best practice, Oracle recommends that you always include version information (an implementation version, or both an implementation and specification version) when creating shared Java EE libraries. Creating and updating version information as you develop shared components allow you to deploy multiple versions of those components simultaneously for testing.

Lesson Review



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Before we move on, let's find out how well you understand the fundamental Java EE and WebLogic concepts and features that have been discussed to this point. Answer this series of simple questions on WebLogic application packaging and configuration.



Place holder for quiz

ORACLE®

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Road Map



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Before we get to the specific techniques used to deploy and manage applications on WebLogic Server, we first need to learn about configuring server resources that applications often depend on. The next lesson in this self-study introduces JDBC server resources, or more specifically WebLogic data sources.

Objectives

After completing this lesson, you should be able to:

- Configure a JDBC data source
- Test and monitor a JDBC data source
- Pool data source connections
- Discuss data source scope

ORACLE

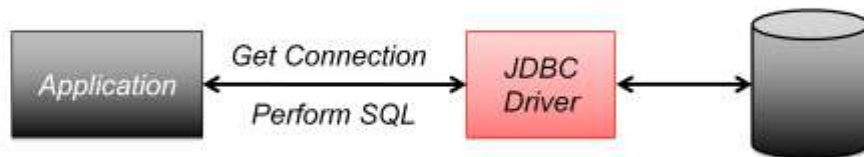
Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In this lesson, you will learn how to create a new JDBC data source using the administration console, followed by how to test and monitor it. You will also learn about using data sources to pool database connections. We'll also use this opportunity to discuss application versus system scoped data source modules.

The *Java Database Connectivity (JDBC)* specification:

- Is a platform- and vendor-independent mechanism for accessing and updating a database
- Provides transparency from proprietary vendor issues
- Requires the use of a *driver*

JDBC drivers are supplied by WLS or your database vendor.



The value of JDBC is that an application can access virtually any data source and run on any platform with a Java Virtual Machine. In other words, with JDBC, it is not necessary to write one program to access a Sybase database, another to access an Oracle database, another to access an IBM DB2 database, and so on. You can write a single program using the JDBC API. Because the application is written in Java, you need not write different applications to run on different platforms, such as Windows and Linux.

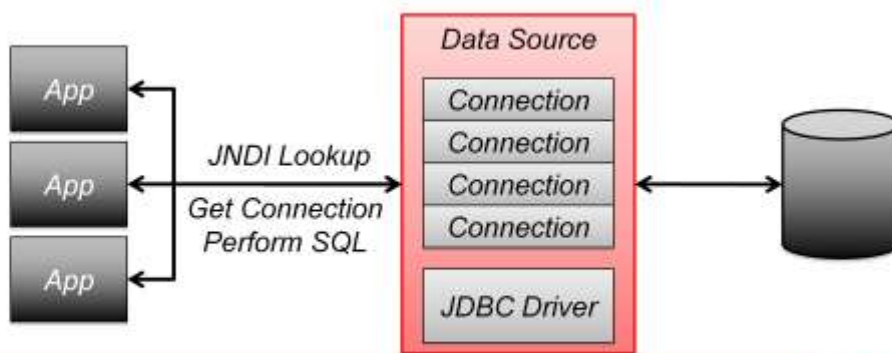
JDBC accomplishes database connections by using a driver mechanism that translates JDBC calls to native database calls. Although most available drivers are fully written in Java (Type 4) and are thus platform independent, some drivers (Type 2) use native libraries and are targeted to specific platforms.

WebLogic Server includes several Type 2 and Type 4 JDBC drivers, which are compliant with the JDBC 3.0 specification. In addition, the Type 4 drivers support the following JDBC 4.0 specification features:

- Connection validation
- Client information storage and retrieval
- Auto-load driver classes (when using Java SE 6)

Data Sources:

- Allow database connectivity to be managed by the application server
- Use a dynamic pool of reusable database connections
- Are obtained by applications from the server's JNDI tree



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

WebLogic Server can manage your database connectivity through JDBC Data Sources and Multi Data Sources. Each data source that you configure contains a pool of database connections that are created when the data source instance is created—when it is deployed or targeted, or at server startup. The connection pool can grow or shrink dynamically to accommodate demand.

Applications look up a data source on the Java Naming and Directory Interface (JNDI) tree or in the local application context (`java:comp/env`), depending on how you configure and deploy the object, and then request a database connection. When finished with the connection, the application uses the close operation on the connection, which simply returns the connection to the connection pool in the data source.

WebLogic data sources allow connection information such as the JDBC driver, the database location (URL), and the username and password to be managed and maintained in a single location, without requiring application to worry about these details. In addition, limiting the number of connections is important if you have a licensing limitation on your database or it can support only a specific capacity.

- Each data source configuration or “module” is persisted as a separate XML document.
- System modules created with the console or WLST are:
 - Stored in the domain's `config/jdbc` directory
 - Available to all applications in the domain
- Application-specific modules are:
 - Deployed as part of Java EE enterprise applications
 - Only accessible by the containing application

Both WebLogic administrators and developers can define JDBC data sources. Regardless of which approach you take, each JDBC data source is represented by an XML file called a module.

WebLogic administrators typically use the Administration Console or the WebLogic Scripting Tool (WLST) to create and deploy (target) JDBC modules. These JDBC modules are considered system modules and are stored in the domain's configuration repository as separate XML files, and are referred to by the domain's `config.xml` file.

Alternatively, developers define data sources in XML descriptor files, and then package the JDBC modules within a Java EE enterprise application for administrators to deploy. These JDBC modules are considered application modules. Because the modules are deployment descriptors, they can also be modified for different environments using Java EE deployment plans.

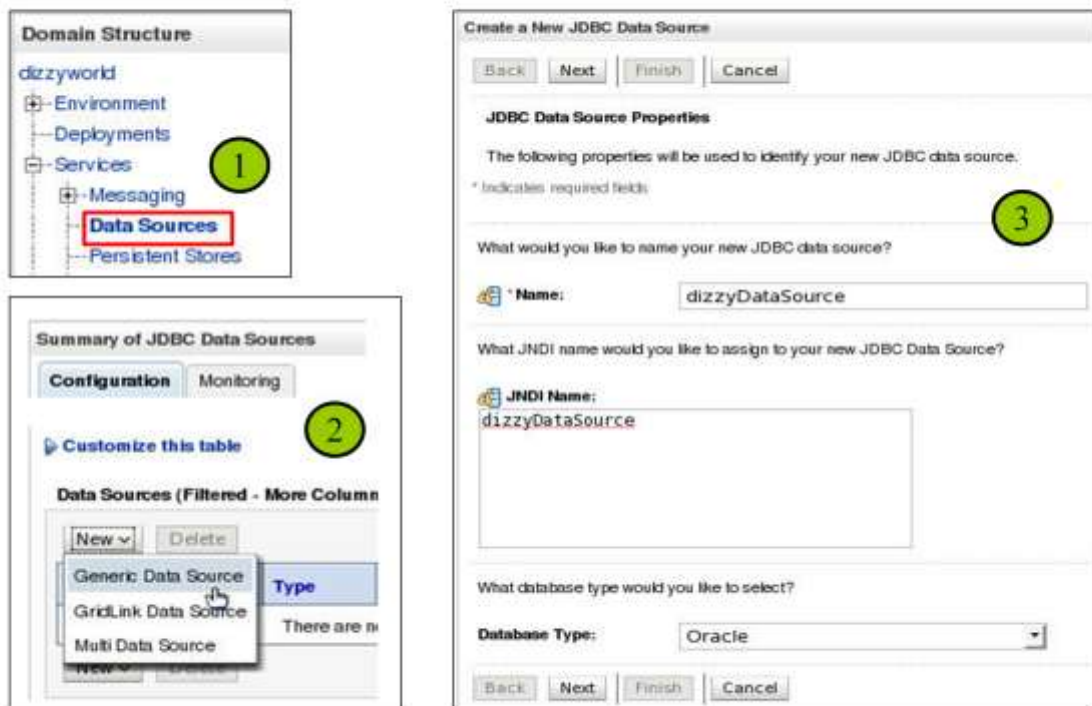
All WebLogic JDBC module files must end with the `-jdbc.xml` suffix, such as `examples-demo-jdbc.xml`. WebLogic Server checks the file name when you deploy the module.

- Oracle and third-party drivers are included in your WLS installation for many popular database products:
 - Oracle 9i, 10g, 11g
 - Sybase Adaptive Server
 - Microsoft SQL Server
 - IBM DB2
 - Informix
 - MySQL
- By default, these drivers are added to your server's classpath.

WebLogic Type 4 JDBC drivers are installed with WebLogic Server in the `WL_HOME\server\lib` folder, where `WL_HOME` is the directory in which you installed WebLogic Server. Driver class files are included in the manifest classpath in `weblogic.jar`, so the drivers are automatically added to your classpath on the server.

The WebLogic Type 4 JDBC drivers are installed by default when you perform a complete installation of WebLogic Server. If you choose a custom installation, ensure that the WebLogic JDBC Drivers option is selected (checked). If this option is deselected, the drivers are not installed.

Create a Data Source



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Make sure that the JDBC drivers that you want to use to create database connections are installed on all servers on which you want to configure database connectivity. Some JDBC drivers are installed with WebLogic Server, including the WebLogic Type 4 JDBC drivers for DB2, Informix, MS SQL Server, Oracle, and Sybase.

1. In the Domain Structure tree, expand `Services` and select `Data Sources`.
2. On the Summary of Data Sources page, click the New drop-down list and select a data source type.
 - **Generic Data Sources:** Provide connection management processes that help keep your system running efficiently.
 - **GridLink Data Sources:** Event-based data sources that adaptively respond to state changes in an Oracle RAC instance.
 - **Multi Data Sources:** Abstraction around a group of generic data sources that provides load balancing or failover processing.

Create a Data Source (continued)

3. On the JDBC Data Source Properties page, enter or select the following information and click Next:
 - **Name:** Enter a configuration name for this JDBC data source.
 - **JNDI Name:** Enter the JNDI name to which this JDBC data source will be bound. Applications needing a database connection look up the data source on the server's JNDI tree.
 - **Database Type:** Select the DBMS of the database that you want to connect to. If your DBMS is not listed, select Other.

Data Source Properties and Transaction Options

Create a New JDBC Data Source

Back Next Finish Cancel

JDBC Data Source Properties

The following properties will be used to identify your new JDBC data source.

Database: Oracle

Type:

What database driver would you like to use to create database connections? Note: * indicates that the driver is explicitly supported by Oracle WebLogic Server.

Database Driver: *Oracle's Driver (Thin XA) for instance connections; Versions: 9.0.1 and later

Back Next Finish Cancel

Create a New JDBC Data Source

Back Next Finish Cancel

Transaction Options

You have selected an XA JDBC driver to use to create database connection in your new data source. The data source will support global transactions and use the 'Two-Phase Commit' global transaction protocol. No other transaction configuration options are available.

Back Next Finish Cancel

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Select the JDBC driver you want to use to connect to the database. The drop-down list includes common JDBC drivers for the selected DBMS.

If you selected an XA driver, the system automatically selects the Two-Phase Commit protocol for global transaction processing.

If you selected a non-XA JDBC driver, local transactions are supported by default. WebLogic Server also offers the option to emulate support for global transactions. If you select the Supports Global Transactions option, you also select which strategy WLS uses to emulate XA with the non-XA driver.

Connection Properties and Test Configuration

Create a New JDBC Data Source

Back Next Finish Cancel

Connection Properties

Define Connection Properties.

What is the name of the database you would like to connect to?

Database Name: orcl

What is the name or IP address of the database server?

Host Name: localhost

What is the port on the database server used to connect to the database?

Port: 1521

What user name do you want to use to create database connections?

Username: scott

What account password to use to create database connections?

Password: *****

Confirm Password: *****

Create a New JDBC Data Source

Test Configuration Next Finish Cancel

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

1. On the Connection Properties page, enter values for the following properties and click Next:
 - **Database Name:** Enter the name of the database that you want to connect to. Exact database name requirements vary by JDBC driver and by DBMS.
 - **Host Name:** Enter the DNS name or IP address of the server that hosts the database.
 - **Port:** Enter the port on which the database server listens for connections requests.
 - **Database User Name:** Enter the database user account that you want to use to connect to the database.
 - **Password/Confirm Password:** Enter the password for the database user account.
2. On the Test Database Connection page, review the connection parameters and click Test Configuration. WebLogic attempts to create a connection from the Administration Server to the database. Results from the connection test are displayed at the top of the page. If the test is unsuccessful, you should correct any configuration errors and retry the test.

Connection Pool Configuration

Settings for dizzyDataSource

Configuration Targets Monitoring

General Connection Pool **1** Oracle

Initial Capacity: 1

Maximum Capacity: 15

Minimum Capacity: 1

Advanced **2**

☐ Test Connections On Reserve

Test Frequency: 120

Shrink Frequency: 900

Connection pool size

Periodically test for bad connections and close

Periodically close idle connections

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

After creating your initial data source configuration in the console, you can tune its connection pool settings:

1. In the Domain Structure tree, expand Services and select Data Sources. Then select your data source and click the Configuration > Connection Pool tab.
2. Enter values for the available data source attributes (some are found under the Advanced options section), including:
 - **Initial Capacity:** The number of physical connections to create when deploying the connection pool. This is also the minimum number of physical connections the connection pool will keep available.
 - **Maximum Capacity:** The maximum number of physical connections that this connection pool can contain.
 - **Minimum Capacity:** The minimum number of physical connections that this connection pool can contain after it is initialized.
 - **Test Frequency:** How often (in seconds) WebLogic Server tests unused connections. Testing requires that you specify a Test Table Name. Connections that fail the test are closed and reopened to reestablish a valid physical connection.
 - **Shrink Frequency:** The number of seconds to wait before shrinking a connection pool that has incrementally increased to meet demand.

Target a Data Source

Deploy data sources to one or more servers in your domain.

Settings for dizzyDataSource

Configuration **Targets** Monitoring Control Security Notes

Save

This page allows you to select the servers or clusters on which you would like to deploy this JDBC data source.

Servers

☐ AdminServer

☒ dizzy3

Clusters

☐ new_Cluster_1

☐ All servers in the cluster

☐ Part of the cluster

☐ dizzy2

☐ dizzy1

Save

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

When you target a JDBC data source, a new instance of the data source is created on the target. When you select a server as a target, an instance of the data source is created on the server. When you select a cluster as a target, an instance of the data source is created on all member servers in the cluster.

1. Navigate to the data source that you want to modify, and click the Targets tab.
2. Select each server or cluster on which you want to deploy the data source, and click Save.

View Server JNDI Tree

Confirm data source deployment using the server's JNDI tree.



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

If the data source was deployed successfully, a new entry should be added to the local JNDI tree of the target servers. The name of the entry should match the JNDI name used to configure the data source. If you used a fully qualified JNDI name containing path separators (for example, "dizzy.datasource.dizzyDataSource"), the entry will not be found at the root of the tree. Instead, directories will be created to match the fully qualified name, if they do not already exist.

1. In the left pane, expand Environment > Servers. Then select a specific server.
2. On the default Configuration > General tab of the server, click View JNDI Tree. The JNDI tree is displayed in a new window.
3. Use the left panel to navigate the directories of the JNDI tree.

Monitor a Data Source

Settings for dizzyDataSource

Configuration Targets **Monitoring** Control Security Notes

Statistics Testing

Monitor data source statistics

Retest data source

[Customize this table](#)

Deployed Instances of this Data Source(Filtered - More Columns Exist)

Showing 1 to 1 of 1 Previous | Next

| State | Connections Total Count | Current Capacity | Waiting For Connection High Count | Highest Num Available | Active Connections Average Count |
|---------|-------------------------|------------------|-----------------------------------|-----------------------|----------------------------------|
| Running | 1 | 1 | 0 | 1 | 0 |

Showing 1 to 1 of 1 Previous | Next

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

After you create a JDBC data source and target it to one or more servers, you can monitor it in the Administration Console. Locate and select your new data source, and click the Monitoring > Statistics tab. Statistics are displayed for each deployed instance of the data source. Optionally, click Customize this table to change the columns displayed in the statistics table.

The available columns include:

Active Connections Current Count: The number of connections currently in use by applications

Active Connections Average Count: Average number of active connections since the data source was deployed

Connections Total Count: The cumulative total number of database connections created in this data source since the data source was deployed

Current Capacity: The current count of JDBC connections in the connection pool in the data source

Highest Num Available: Highest number of database connections that were available at any time in this instance of the data source since the data source was deployed

Lesson Review



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

It's time once again to find out how well you understand the fundamental WebLogic concepts and features that have been discussed in this lesson. Answer this series of simple questions on JDBC data sources.



Place holder for quiz

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Road Map



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Another type of server resource that is commonly required by Java EE applications is the Java Messaging Service, or JMS. The next lesson in this course introduces you to the concepts, architecture, and basic configuration process of JMS on WebLogic Server.

Objectives

After completing this lesson, you should be able to:

- Describe WebLogic's JMS architecture
- Compare JMS queues to JMS topics
- Configure JMS servers, destinations and connection factories
- Configure persistent messaging
- Monitor JMS resources

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

First, this lesson reviews the overall JMS architecture and terminology, including the concepts of message queues and message topics. We'll then learn how to use the console to configure JMS resources on WebLogic, such as JMS servers, JMS destinations, and JMS connection factories. The lesson also introduces the concept of persistent messaging to help achieve high availability for JMS. Finally, just as we did with data sources, we'll see how the console can be used to monitor active JMS resources.

The Java Message Service (JMS):

- Is a standard Java API and underlying protocol
- Allows Java EE applications to communicate with message-oriented middleware
- Promotes application integration in a loosely-coupled fashion
- Supports both point-to-point and publish-subscribe messaging models



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

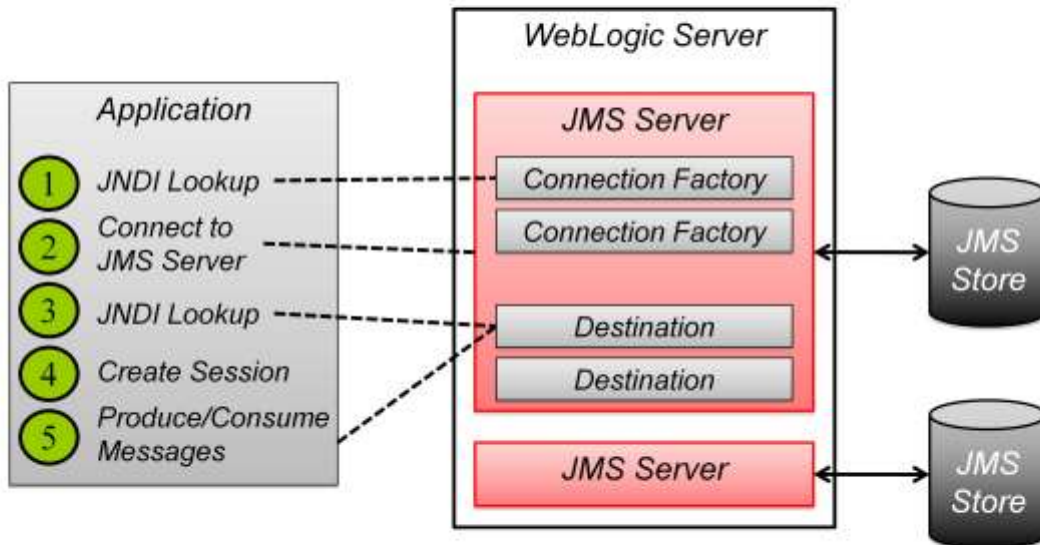
Message-oriented middleware enables applications to communicate with one another through the exchange of messages. A message is a request, report, and/or event that contains information needed to coordinate communication between different applications. A message provides a level of abstraction, allowing you to separate the details about the destination system from the application code.

The Java Message Service (JMS) is a standard API for accessing enterprise messaging systems. Specifically, JMS enables Java applications to share a messaging system to exchange messages, and also simplifies application development by providing a standard interface for creating, sending, and receiving messages.

WebLogic JMS accepts messages from producer applications and delivers them to consumer applications. JMS is tightly integrated into the WebLogic Server platform, allowing you to build highly secure Java EE applications that can be easily monitored and administered through the WebLogic Server console. In addition, WebLogic JMS supports XA transactions as well as high availability through persistent stores and clustering.

JMS supports two messaging models: point-to-point (PTP) and publish/subscribe (pub/sub). The terms producer and consumer are used as generic descriptions of applications that send and receive messages, respectively, in either messaging model. For each specific messaging model, however, unique terms specific to that model are used when referring to producers and consumers.

WebLogic JMS Architecture



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

JMS servers host a defined set of JMS modules and any associated persistent storage on a WebLogic Server instance. JMS modules contain one or more JMS resources such as message destinations and connections factories. Client JMS applications either produce messages to destinations or consume messages from these destinations. A typical JMS interaction between a client application and a server involves the following steps:

1. Clients use the server's JNDI (Java Naming and Directory Interface) service to look up and download a connection factory.
2. Clients use the connection factory to establish a new connection to the JMS server.
3. Clients use JNDI to look up one or more JMS destinations on which to send and/or receive messages.
4. Clients start a new JMS session with the server. Sessions can be used to, among other things, send and receive multiple messages within an atomic transaction.
5. Clients produce messages and send them to a destination, and/or consume messages and receive them from a destination.

Create a JMS Server



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

A JMS server's primary responsibility for its destinations is to maintain information on what persistent store is used for any persistent messages that arrive on the destinations, and to maintain the states of durable subscribers created on the destinations. JMS servers also manage message paging on destinations, and, optionally, can manage message and/or byte thresholds, as well as server-level quota for its targeted destinations. As a container for targeted destinations, any configuration or run-time changes to a JMS server can affect all the destinations that it hosts.

1. In the Administration Console, expand **Services > Messaging** and select **JMS Servers**. On the Summary of JMS Servers page, click **New**.
2. Enter a name for the JMS server and click **Next**.
3. On the Selects Targets page, select the server instance on which to deploy the JMS server.

- JMS destinations and connection factories are packaged and deployed as one or more *JMS Modules*:
 - Persisted as a separate XML documents
 - Targeted to whole servers or specific JMS servers
- System modules are:
 - Created with the console or WLST
 - Stored in the domain's `config/jms` directory
- Application-specific modules are deployed as part of Java EE enterprise applications.

WebLogic Administrators typically use the Administration Console or the WebLogic Scripting Tool (WLST) to create and deploy (target) JMS modules, and to configure the module's configuration resources, such as queues, and topics connection factories. JMS modules that you configure this way are considered system modules. JMS system modules are owned by the Administrator, who can at any time add, modify, or delete resources. System modules are globally available for targeting to servers and clusters configured in the domain and, therefore, are available to all applications deployed on the same targets and to client applications. When you create a JMS system module, WebLogic Server creates a JMS module file in the `config\jms` subdirectory of the domain directory, and adds a reference to the module in the domain's `config.xml` file as a `JMSSystemResource` element. This reference includes the path to the JMS system module file and a list of target servers and clusters on which the module is deployed.

JMS configuration resources can also be managed as deployable application modules, similar to standard Java EE descriptor-based modules. JMS Application modules can be deployed either with a Java EE application as a packaged module, where the resources in the module are optionally made available to only the enclosing application (that is, application-scoped), or as a stand-alone module that provides global access to the resources defined in that module.

Create a JMS Module

Domain Structure

- dizzyworld
 - Environment
 - Deployments
 - Services
 - Messaging
 - JMS Servers
 - Store-and-Forward Agents
 - JMS Modules** (1)
 - Path Services

Create JMS System Module

Back Next Finish Cancel

The following properties will be used to identify your new module.

JMS system resources are configured and stored as modules similar to standard servers, and JMS store-and-forward (SAF) parameters. You can administratively

* Indicates required fields

What would you like to name your System Module?

* Name: dizzyworldModule (2)

Create JMS System Module

Back Next Finish Cancel

The following properties will be used to target your new JMS system module.

Use this page to select the server or cluster on which you would like to deploy this JMS

Targets: (3)

Servers

- ☐ AdminServer
- ☐ dizzy3

Clusters

- ☒ new_Cluster_1
 - ☐ All servers in the cluster
 - ☒ Part of the cluster
 - ☐ dizzy2
 - ☒ dizzy1

Back Next Finish Cancel

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

JMS resources are configured and stored as modules similar to standard J2EE modules. Such resources include queues, topics, connection factories, templates, destination keys, quota, distributed queues, distributed topics, foreign servers, and JMS store-and-forward (SAF) parameters. You can administratively configure and manage JMS modules as global system resources.

1. In the Administration Console, expand Services > Messaging and select JMS Modules. On the Summary of JMS Modules page, click New.
2. On the Create JMS System Module page, enter values for the following and click Next:
 - **Name:** Enter a name for the JMS system module.
 - **Descriptor File Name:** Optionally enter a name for the module's underlying descriptor file. The system automatically adds a "-jms.xml" extension to this name. If you do not provide a name, a default name will be assigned.
3. On the Targets page, select the server instance or cluster target on which to deploy the JMS system module, and click Next. Click Finish.

Deploy a Module to a JMS Server

The image displays four screenshots from the Oracle Administration Console, illustrating the process of deploying a module to a JMS server. The steps are numbered 1 through 4.

- Step 1:** A screenshot of the **Domain Structure** tree. The path is **Services > Messaging > JMS Servers > Store-and-Forward > JMS Modules**. The **JMS Modules** node is highlighted with a red box and a green circle with the number 1.
- Step 2:** A screenshot of the **Settings for dizzyworldModule** page. The **Subdeployments** tab is selected, and the **New** button is highlighted with a green circle with the number 2.
- Step 3:** A screenshot of the **Create a New Subdeployment** page. The **Subdeployment Properties** section is shown, with the **Subdeployment Name** field containing the text **dizzysubmodule**. A green circle with the number 3 is next to the **Next** button.
- Step 4:** A screenshot of the **Create a New Subdeployment** page, showing the **Targets** section. The **JMS Servers** section is expanded, and the **dizzyworldJMSServer** checkbox is checked. A green circle with the number 4 is next to the **Finish** button.

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

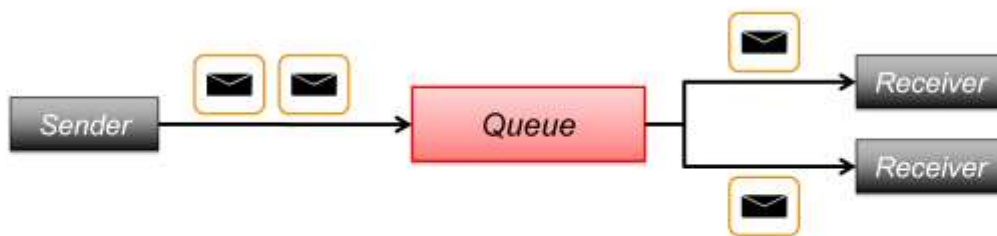
Because destination resources are encapsulated in JMS modules, they are not nested under JMS servers in the configuration file. However, a subtargeting relationship between JMS servers and destinations is maintained because each stand-alone destination resource within a JMS module is always targeted to a single JMS server. This way, JMS servers continue to manage persistent messages, durable subscribers, message paging, and, optionally, quotas for their targeted destinations. Multiple JMS modules can be targeted to each JMS server in a domain.

1. In the Administration Console, expand **Services > Messaging**, and select **JMS Modules**. Then select your existing JMS module.
2. Click the **Subdeployments** tab, and click the **New** button.
3. On the **Subdeployment Properties** page, enter a name for the subdeployment. Click **Next**.
4. In the **JMS Servers** section, select each JMS server instance on which you want to deploy the resources associated with the subdeployment. Click **Finish**.

Queue Destinations

In JMS *point-to-point* messaging:

- Clients communicate with a *queue* destination
- Messages are distributed to consumers in a serial fashion (first in, first out)
- Each message is only delivered to a single consumer



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

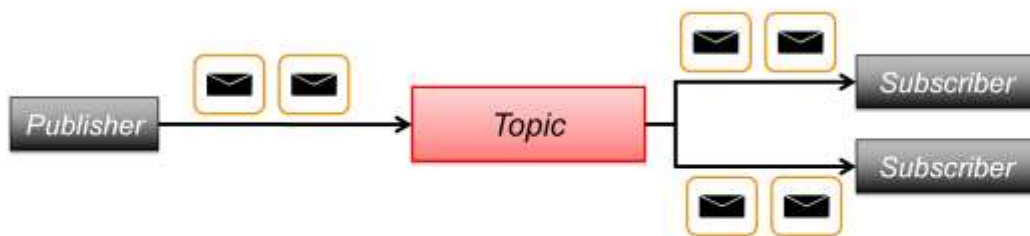
The point-to-point (PTP) messaging model enables one application to send a message to another. PTP messaging applications send and receive messages using named queues. A queue sender (producer) sends a message to a specific queue. A queue receiver (consumer) receives messages from a specific queue. Multiple queue senders and queue receivers can be associated with a single queue, but an individual message can be delivered to only one queue receiver.

If multiple queue receivers are listening for messages on a queue, WebLogic JMS determines which one will receive the next message on a first come, first serve basis. If no queue receivers are listening on the queue, messages remain in the queue until a queue receiver attaches to the queue.

Topic Destinations

In JMS *publish/subscribe* messaging:

- Clients communicate with a *topic* destination
- Messages are broadcast to all subscribers
- A message can be saved until at least one subscriber has consumed it ("durable")



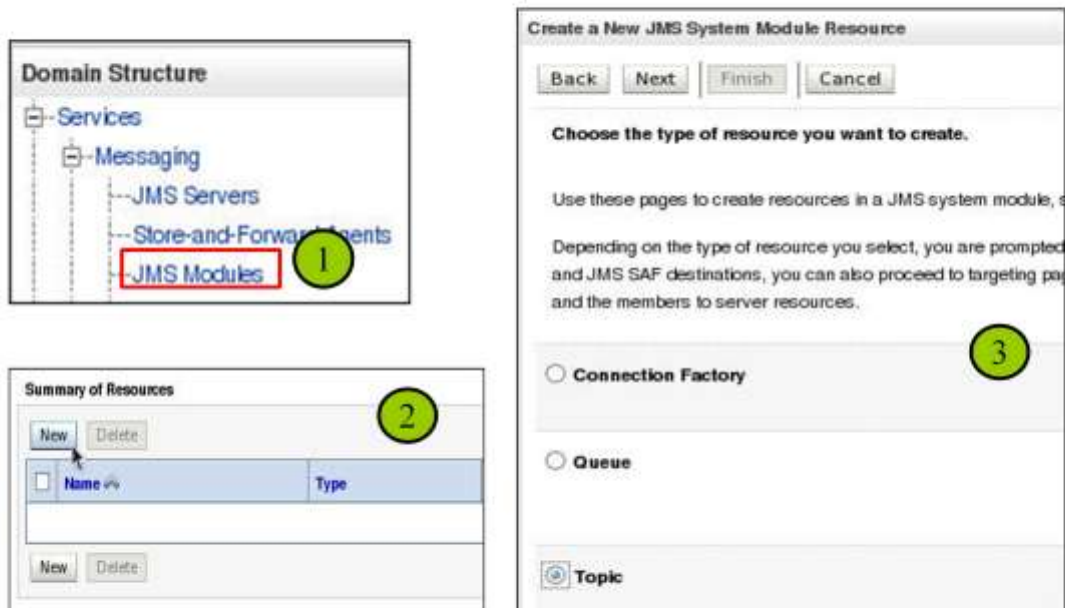
ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The publish/subscribe (pub/sub) messaging model enables an application to send a message to multiple applications. Pub/sub messaging applications send and receive messages by subscribing to a topic. A topic publisher (producer) sends messages to a specific topic. A topic subscriber (consumer) retrieves messages from a specific topic. Unlike with the PTP messaging model, the pub/sub messaging model allows multiple topic subscribers to receive the same message. JMS retains the message until all topic subscribers have received it.

The Pub/Sub messaging model supports durable subscribers. For durable subscriptions, WebLogic JMS stores a message in a persistent file or database until the message has been delivered to the subscribers or has expired, even if those subscribers are not active at the time that the message is delivered. To support durable subscriptions, a client identifier (client ID) must be defined for the connection by the JMS client application. Support for durable subscriptions is a feature unique to the Pub/Sub messaging model, so client IDs are used only with topic connections; queue connections also contain client IDs, but JMS does not use them.

Create a Destination



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

After creating a JMS system module, you can configure resources for the module, including stand-alone queues and topics, distributed queues and topics, connection factories, JMS templates, destination sort keys, destination quota, foreign servers, and JMS SAF (store-and-forward) parameters.

For each destination, you can optionally select a (or create a new) subdeployment for the resource. A subdeployment is a mechanism by which targetable JMS module resources (such as queues, topics, and connection factories) are grouped and targeted to a server resource (such as JMS servers, server instances, or a cluster).

1. In the Administration Console, expand Services > Messaging and select JMS Modules. Select an existing JMS module.
2. On the Configuration page, click New above the Summary of Resources table.
3. Select the type of destination to create: Queue or Topic. Click Next.

Create a Destination

Create a New JMS System Module Resource

Back Next Finish Cancel

JMS Destination Properties

The following properties will be used to identify your new Topic. The current module is dizzyworldModule.

* Indicates required fields:

* Name: dizzyworldTopic

JNDI Name: dizzyworldTopic

Template: None

Back Next Finish Cancel

Create a New JMS System Module Resource

Back Next Finish Cancel

The following properties will be used to target your new JMS system module resource

Use this page to select a subdeployment to assign this system module resource. A subdeployment resources are grouped and targeted to a server instance, cluster, or SAF agent. If necessary, you can create a new subdeployment by clicking the **Create a New Subdeployment** button. You can also reconfigure subdeployment targeting on the subdeployment management page.

Select the subdeployment you want to use. If you select (none), no targeting will occur.

Subdeployments: dizzysubmodule Create a New Subdeployment

1. Enter Name and JNDI Name for the destination. Click Next.
2. Select an existing subdeployment from this JMS module. Your new JMS destination will be targeted to the JMS servers indicated by the subdeployment.

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

- *JMS Connection Factories* are used to set default client connection parameters, including:
 - Message priority
 - Message time-to-live (TTL)
 - Message persistence
 - Transactional behavior
 - Acknowledgement policy
 - Flow control
- WLS provides a default client connection factory that:
 - Uses WebLogic's default connection settings
 - Is located on the server JNDI tree at `weblogic.jms.ConnectorFactory`

Connection factories are resources that enable JMS clients to create JMS connections. A connection factory supports concurrent use, enabling multiple threads to access the object simultaneously. WebLogic JMS provides preconfigured default connection factories that can be enabled or disabled on a per-server basis. Otherwise, you can configure one or more connection factories to create connections with predefined options that better suit your application.

Some connection factory options are dynamically configurable. When options are modified at run time, only incoming messages are affected; stored messages are not affected.

You can modify the following parameters for connection factories:

- General configuration parameters, including modifying the default client parameters, default message delivery parameters, load-balancing parameters, unit-of-order parameters, and security parameters
- Transaction parameters, which enable you to define a value for the transaction time-out option and to indicate whether an XA queue or XA topic connection factory is returned, and whether the connection factory creates sessions that are XA aware
- Flow control parameters, which enable you to tell a JMS server or destination to slow down message producers when it determines that it is becoming overloaded

Create a Connection Factory

Domain Structure

- Services
 - Messaging
 - JMS Servers
 - Store-and-Forward Agents
 - JMS Modules** (1)

Create a New JMS System Module Resource

Back Next Finish Cancel

Choose the type of resource you want to create.

Use these pages to create resources in a JMS system module, such as queues, topics, and JMS SAF destinations. Depending on the type of resource you select, you are prompted to enter basic information and JMS SAF destinations, you can also proceed to targeting pages for selecting appropriate members to server resources.

☒ **Connection Factory** (2)

☐ Queue

Create a New JMS System Module Resource

Back Next Finish Cancel

Connection Factory Properties (3)

The following properties will be used to identify your new connection factory. The current mode is **Basic**.

* Indicates required fields:

What would you like to name your new connection factory?

Name: dizzyworldConnection

What JNDI Name would you like to use to look up your new connection factory?

JNDI Name: dizzyworldConnection

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Within each JMS module, connection factory resource names must be unique. And, all connection factory JNDI names in any JMS module must be unique across an entire WebLogic domain.

1. In the Administration Console, expand Services > Messaging, and select JMS Modules. Select an existing JMS module.
2. Select the Connection Factory resource type and click Next.
3. Enter Name and JNDI Name for the new connection factory, and click Next.
4. (not shown) For basic default targeting, accept the default targets presented in the Targets box, and then click Finish. For advanced targeting, click Advanced Targeting, which allows you to select an existing subdeployment or to create a new one. When a valid subdeployment is selected, its targeted JMS servers, servers, or cluster appear as selected in the Targets box.

Configure a Connection Factory

Settings for dizzyworldConnection

Configuration Subdeployment Notes

General **Default Delivery** Client Transactions Flow Control Load Balance Security

Save

Use this page to define the default delivery configuration parameters for this JMS connection factory, such as the default priority, time-to-live, and time-to-deliver.

Default Priority: 4

Default Time-to-Live: 0

Default Time-to-Deliver: 0

Default Delivery Mode: Persistent

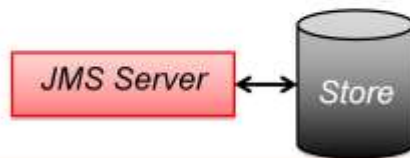
ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In the Administration Console, navigate to the connection factory resource that you want to configure. Click the Configuration > Default Delivery tab.

- **Default Priority:** The default priority used for messages when a priority is not explicitly defined. Values are 0–9.
- **Default Time-to-Live:** The maximum length of time, in milliseconds, that a message will exist. This value is used for messages when a priority is not explicitly defined. A value of 0 indicates that the message has an infinite amount time to live.
- **Default Time-to-Deliver:** The delay time, in milliseconds, between when a message is produced and when it is made visible on its destination
- **Default Delivery Mode:** Whether or not messages should use a persistent store, if one is associated with the JMS server
- **Default Redelivery Delay:** The delay time, in milliseconds, before rolled back or recovered messages are redelivered
- **Send Timeout:** The maximum length of time, in milliseconds, that a sender will wait when there is not enough available space (no quota) on a destination to accommodate the message being sent. The default time is 10 milliseconds.

- WebLogic supports guaranteed messaging using Persistent Stores:
 - In-progress messages can be delivered despite server restart
 - Topic subscribers can consume missed messages despite reconnecting to the server
- The following types of JMS Persistent Stores are available:
 - File system
 - JDBC (requires an existing Data Source)

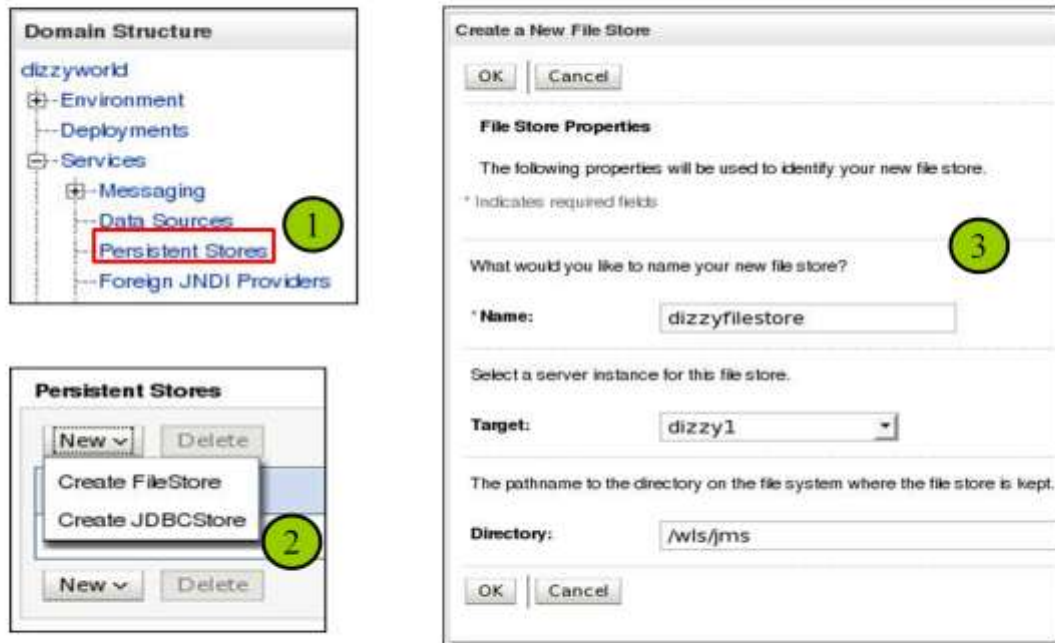


A persistent message is guaranteed to be delivered only once. The message cannot be lost due to a JMS provider failure and it must not be delivered twice. It is not considered sent until it has been safely written to a WebLogic persistent store that has been assigned to each JMS server during configuration.

Nonpersistent messages are not stored. They are guaranteed to be delivered at most once, unless there is a JMS provider failure, in which case messages may be lost, and must not be delivered twice. If a connection is closed or recovered, all nonpersistent messages that have not yet been acknowledged will be redelivered. After a nonpersistent message is acknowledged, it will not be redelivered.

WebLogic Persistent Stores provide built-in, high-performance storage solutions for WebLogic Server subsystems and services that require persistence. For example, it can store persistent JMS messages or temporarily store messages sent using the JMS Store-and-Forward feature. The persistent store supports persistence to a file-based store or to a JDBC-enabled database. Each server instance, including the administration server, has a default persistent store that requires no configuration. The default store is a file-based store that maintains its data in a group of files in a server instance's `data\store\default` directory.

Create a JMS Store



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

You may want to create a custom file store so that the file store's data is persisted to a particular storage device or when you want a JMS service that accesses a file store to be able to migrate with the store to another server member in a cluster. When configuring a file store directory, the directory must be accessible to the server instance on which the file store is located.

1. In the left pane of the console, expand Services and select Persistent Stores.
2. On the Summary of Persistent Stores page, select the store type from the New drop-down list.
3. If File Store is selected, update the following on the Create a new File Store page:
 - **Name:** Name of the store
 - **Target:** Server instance on which to deploy the store
 - **Directory:** Pathname to the directory on the file system where the file store is kept. This directory must exist on your system, so be sure to create it before completing this tab. For highest availability, use either a Storage Area Network (SAN) or a dual-ported SCSI disk.

Assign a Store to a JMS Server



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Associate your new custom persistent store with a JMS server, using the Persistent Store field of the Configuration > General tab. If you leave this field set to none, the JMS server uses the default file store that is automatically configured on each targeted server instance.

Monitor JMS Servers

Settings for dizzyworldJMServer

Configuration Logging Targets **Monitoring** Control Notes

Monitoring Paging Store Active Destinations Active Transactions Active Connections Active Session Pools

Active Pooled Connections

Use this page to view runtime statistics for all of the active JMS servers, active destinations, active transactions, session pools in the current domain.

[Customize this table](#)

Statistics (Filtered - More Columns Exist)

Showing 1 to 1 of 1 Previous Next

| Name | Destinations Current | Messages Current | Messages Pending | Messages Received | Messages Pageable Current | Messages Paged Out Total | Bytes Current |
|--------------------|----------------------|------------------|------------------|-------------------|---------------------------|--------------------------|---------------|
| dizzyworldJMServer | 1 | 2 | 0 | 2 | 0 | 0 | 20 |

Showing 1 to 1 of 1 Previous Next

Domain Structure

- dizzyworld
 - Environment
 - Deployments
 - Services
 - Messaging**
 - JMS Servers**
 - Store-and-Forward
 - JMS Modules

You can monitor statistics for view runtime information for an active JMS server. From here, you can also access runtime information for a JMS server's destinations, transactions, connections, and server session pools.

Expand Services > Messaging and select JMS Servers. Select a JMS server.

Click the Monitoring tab. By default a Monitoring second level tab is displayed, which provides general statistics for all destinations on every JMS server in the domain. These statistics include the number and size of messages processed by the JMS server.

The Active Destinations tab displays statistics for each active JMS destinations for the domain.

The Active Transactions tab displays all active JMS transactions for the domain. For troubleshooting purposes, you can force commits or rollbacks on selected transactions. Simply select a transaction, and then click either the Force Commit or Force Rollback button.

The Active Connections tab displays all active client JMS connections for the domain. For troubleshooting purposes, you can destroy selected connections. Simply select a connection, and then click the Destroy button above the table.

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Monitor & Manage Destinations

Settings for dizzyworldJMServer

Configuration Logging Targets **Monitoring** Control Notes

Monitoring Paging Store **Active Destinations** Active Transactions Active Connections Active Session Pools

Active Pooled Connections

This page allows you to view active destinations targeted to this JMS server.

[Customize this table](#)

Destinations (Filtered - More Columns Exist)

Production Consumption Insertion

Showing 1 to 2 of 2 Previous | Next

| <input type="checkbox"/> | Name | Messages Current | Messages Pending | Messages High | Messages Received | Messages Threshold | Destination Type |
|--------------------------|----------------------------------|------------------|------------------|---------------|-------------------|--------------------|------------------|
| <input type="checkbox"/> | dizzyworldModule\dizzyqueue | 2 | 0 | 2 | 2 | 0 | Queue |
| <input type="checkbox"/> | dizzyworldModule\dizzyworldTopic | 0 | 0 | 0 | 0 | 0 | Topic |

Production Consumption Insertion

Showing 1 to 2 of 2 Previous | Next

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

For troubleshooting purposes, you can temporarily pause all run-time message production, insertion (in-flight messages), and consumption operations on any or all destinations targeted to the selected JMS server. These "message pausing" options allow you to assert administrative control of the JMS subsystem behavior in the event of an external resource failure.

The available columns include:

Messages Current: The current number of messages in the destination. This does not include the pending messages.

Messages Pending: The number of pending messages in the destination. A pending message is one that has either been sent in a transaction and not committed, or that has been received and not committed or acknowledged.

Messages High: The peak number of messages in the destination since the last reset.

Messages Received: The number of messages received in this destination since that reset.

Messages Threshold: The amount of time in the threshold condition since the last reset.

Consumers Current: The current number of consumers accessing this destination.

Lesson Review



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Let's see how well you understand the basic architecture and terminology associated with WebLogic's JMS features. Answer the following series of simple questions.



Place holder for quiz

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Road Map



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In the final section of this course, we'll cover the different methods that you can use to install, deploy, and redeploy applications in WebLogic Server.

Objectives

After completing this lesson, you should be able to:

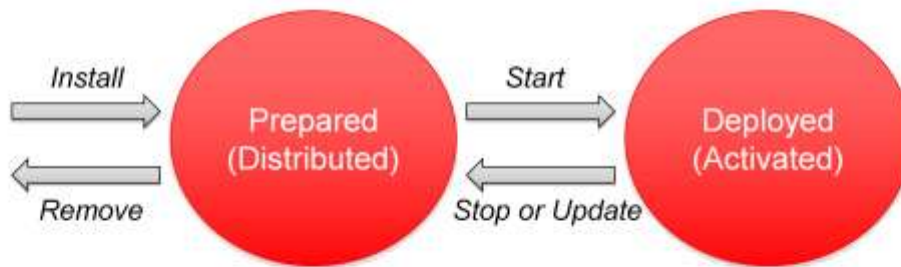
- List several options for deploying applications to WebLogic Server
- Use the console to install a new application or update an existing one
- Associate an application with a deployment plan
- Discuss WebLogic's production redeployment features
- Monitor application statistics

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

After this lesson you'll be able to describe in detail the different deployment options available to WebLogic administrators. Of course this includes the use of the administration console to install and update applications. We'll also use this opportunity to introduce some advanced WebLogic deployment techniques, including Java EE deployment plan files and the production redeployment features. Finally, this lesson wraps up with a discussion of basic application monitoring.

Application Life Cycle



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

WebLogic applications go through several deployment states. First, a newly installed application enters the Prepared phase. During this phase, the application files are distributed to the initial target servers in the domain. By default, WebLogic stages applications to remote managed servers by uploading them over a network. Alternatively, WebLogic can stage applications to a local directory that is shared amongst the other managed servers. From the Prepared phase the application can transition to the Deployed phase, in which the application is activated on all target servers and is ready to accept client requests.

Not shown here is an additional Administrative phase. Applications in this phase are also active, but can only receive requests from the domain's administrative network channel, if configured. The Administrative phase can be useful if you would like to verify successful deployment before allowing client traffic.

Several methods are available to deploy WebLogic applications and shared libraries, including:

- Administration console
- WebLogic Scripting Tool (WLST)
- `weblogic.Deployer` Java class
- `wldeploy` Ant task
- Auto-deployment folder

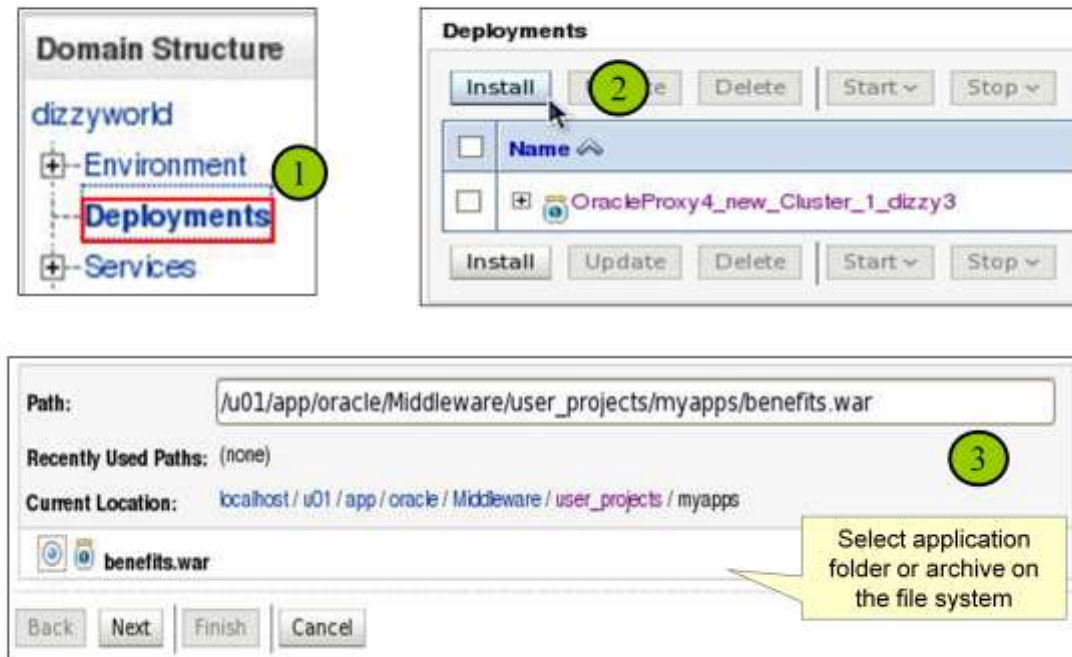
The Administration Console provides a series of Web-based Deployment Assistants that guide you through the deployment process. The Administration Console also provides controls for changing and monitoring the deployment status, and changing selected deployment descriptor values while the deployment unit is up and running. Use the Administration Console when you need to perform basic deployment functions interactively and you have access to a supported browser.

The WebLogic Scripting Tool (WLST) is a scripting interface that you can use to automate domain configuration tasks, including application deployment configuration and deployment operations.

Similarly, `weblogic.Deployer` provides a command-line based interface for performing both basic and advanced deployment tasks. Use `weblogic.Deployer` when you want command-line access to WebLogic Server deployment functionality, or when you need to perform a deployment task that is not supported using the Administration Console. The `wldeploy` task is an Ant-based version of the `weblogic.Deployer` utility. You can automate deployment tasks by placing `wldeploy` commands in an Ant `build.xml` file and running Ant to execute the commands.

The `autodeploy` domain directory allows you to deploy an application quickly for evaluation or testing in a development environment.

Prepare a New Application



ORACLE

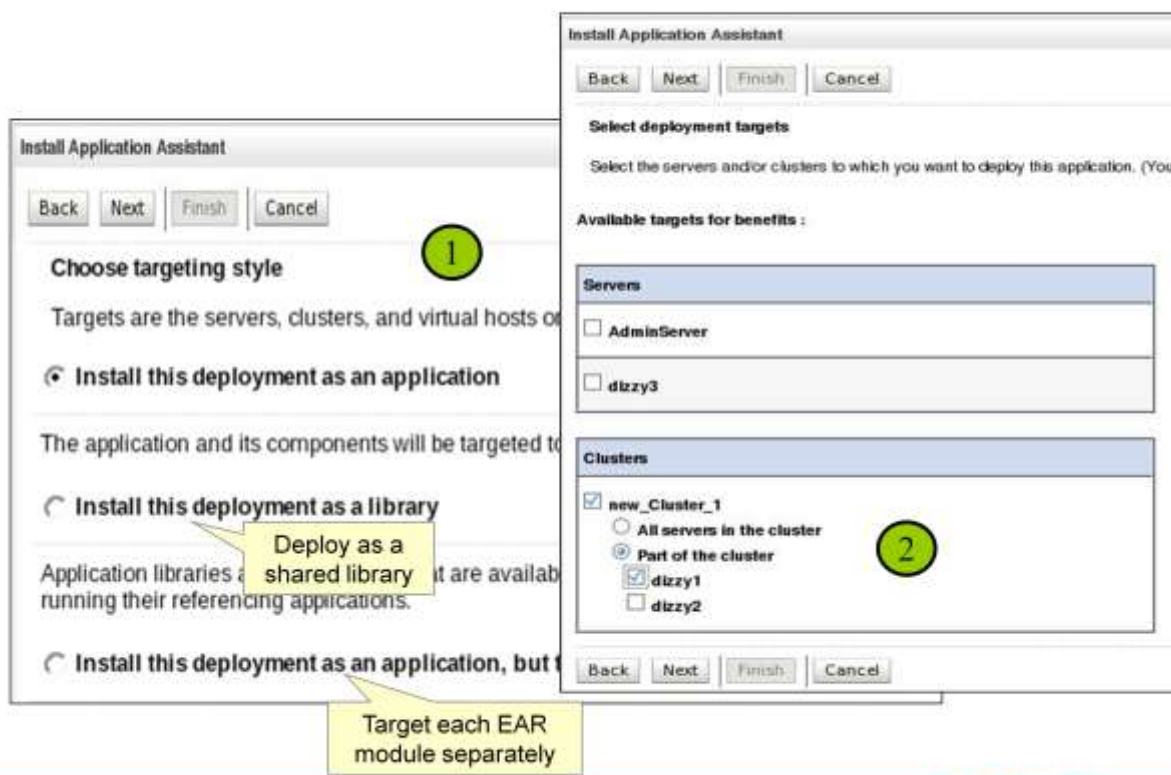
Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

All types of Java EE applications are deployed in the same way in WebLogic Server. Installing an application refers to making its physical file or directory known to WebLogic Server. An application can be installed as an archived file or as an exploded directory. After you have installed the application, you can start it so that users can begin using it.

1. In the left pane of the Console, select Deployments.
2. In the right pane, click Install.
3. Locate the file or exploded directory that corresponds to the application you want to install. Either enter the path on the local file system manually, or use the Current Location or Recently Used Paths features. If using these features, select the path or file using the supplied option buttons. Click Next.

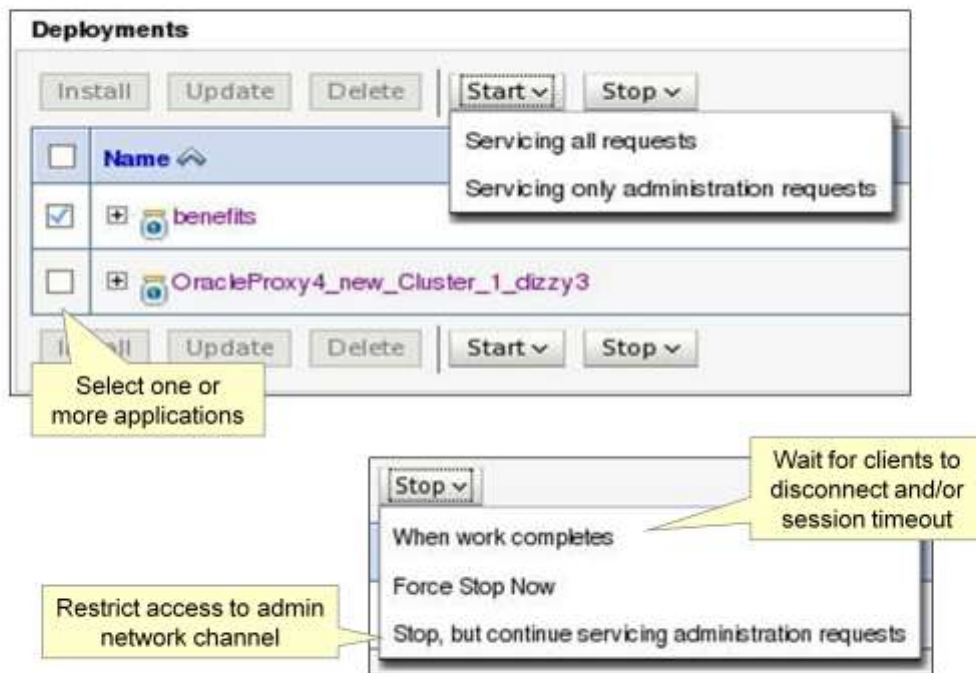
If the application archive is not accessible from the administration server's file system, you can use the Upload link to upload the application from your browser's file system to the path indicated by Current Location.

Prepare a New Application



1. Select **Install this deployment as an application** to register it as a client-accessible application. Select **Install this deployment as a library** to instead register it as a shared library that can be referenced by other applications. If you have created additional managed servers or clusters and you are installing an Enterprise application, you can also select **Install this deployment as an application, but target the modules individually**. Select this option if, for example, you want to target all Web modules in the Enterprise application to one managed server and all EJB modules to a different managed server. Click **Next**.
2. Select the managed servers or clusters to which you want to deploy the application. Click **Next**.
3. (not shown) Optionally update settings about the deployment, including the configuration name, and the security model to use (deployment descriptor and/or WebLogic roles and policies). Click **Next**.
4. (not shown) Review the configuration settings you have chosen, specify whether you want to immediately update the application's configuration after you install it, then click **Finish** to complete the installation.

Deploy or Undeploy Applications



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

When you start an application, you can make it immediately available to clients, or you can start it in Administration mode to first ensure that it is working. Starting in Administration mode allows you to perform the final ("sanity") checking of the distributed application directly in the production environment without disrupting clients. Administration mode restricts access to an application to a configured Administration channel. Similarly, you can stop an application so that no clients can use it, or you can stop it in Administration mode so that only the administrative tasks can be performed.

Stopping an application does not remove its source files from the server; you can later redeploy (also called update) a stopped application to make it available to WebLogic Server clients once again.

To deploy or undeploy applications:

1. In the left pane of the Console, select Deployments. A table in the right pane displays all deployed applications and modules.
2. Select the check boxes for the applications you want to start or stop.
3. Click the Start or Stop buttons, and select from the available options.

Redeploy an Application



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

When you update an application, you can specify that WebLogic Server redeploy the original archive file or exploded directory, or you can specify that WebLogic Server deploy a new archive file in place of the original one. You can also change the directory that contains the deployment plan that is associated with the application. Update an application if you have made changes to the application and you want to make the changes available to WebLogic Server clients, or if you want to redeploy an entirely new archive file in a new location.

1. In the left pane of the Console, select Deployments. Select the check boxes for the applications you want to redeploy. Click the Update button.
2. If the path of the application has not changed, click Finish. Otherwise, use the Change Path button to select a new location. If your application uses a deployment plan, you also have the option of changing its location as well using the Change Plan button.

Deployment with WLST

Prepare and deploy a new application, or redeploy an existing one:

```
connect('myuser','mypass','t3://adminserver:7001')
name = "HRServices"
location = "/usr/myapplications/HRServices.ear"

deploy(name, location, targets='serverA')
```

Other WLST deployment commands:

```
distributeApplication(location, targets='serverA')
startApplication(name)
redeploy(name)
stopApplication(name)
listApplications()
```

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Available WLST deployment commands include:

- **deploy**: Deploys an application or library to a WebLogic Server instance, and returns a WLSTProgress object that you can access to check the status of the command. Many options are supported, including the staging mode, version, deployment plan, and various timeouts. You can also use this command to upload the application remotely to the administration server's file system.
- **distributeApplication**: Copies the deployment bundle to the specified targets. The deployment bundle includes module, configuration data, and any additional generated code. The `distributeApplication` command does not start deployment.
- **startApplication**: Starts an application, making it available to users. The application must be fully configured and available in the domain.
- **redeploy**: Reloads classes and redeploys a previously deployed application
- **stopApplication**: Stops an application, making it unavailable to users. The application must be fully configured and available in the domain.
- **listApplications**: Lists all applications that are currently deployed in the domain
- **undeploy**: Undeploys an application from the specified servers

Deployment with weblogic.Deployer

Prepare and deploy a new application:

```
java weblogic.Deployer -adminurl t3://adminserver:7001  
-username myuser -password mypass -name HRServices  
-source /usr/HRServices.ear -targets serverA -deploy
```

Redeploy an application:

```
java weblogic.Deployer -adminurl t3://adminserver:7001  
-username myuser -password mypass -name HRServices  
-redeploy
```

Undeploy an application:

```
java weblogic.Deployer -adminurl t3://adminserver:7001  
-username myuser -password mypass -name HRServices  
-undeploy
```

List all applications:

```
java weblogic.Deployer -adminurl t3://adminserver:7001  
-username myuser -password mypass -listapps
```

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

You can use the `setWLSEnv.sh` or `setWLSEnv.cmd` script, located in the `server/bin` subdirectory of the WebLogic Server installation directory, to set the required environment to run `weblogic.Deployer`. The general syntax for invoking `weblogic.Deployer` is the following:

```
java [SSL Arguments] weblogic.Deployer [Connection Arguments] [User  
Credentials Arguments] COMMAND-NAME command-options [Common  
Arguments]
```

Available commands include:

- **deploy:** Deploys or redeploys an application, library or module. Specify the application archive or path along with a comma-separated list of targets. Many options are supported, including the staging mode, version, deployment plan, and various timeouts. You can also use this command to upload the application remotely to the administration server's file system.
- **distribute:** Prepares deployment files for deployment by copying deployment files to target servers and validating them. A distributed application can be started quickly with the start command.
- **redeploy:** Redeploys a running application or part of a running application
- **start:** Makes a stopped (inactive) application available to clients on target servers. The start command does not redistribute deployment files to target servers.

- By default, the *Auto-Deployment* feature is only enabled if the domain is **not** running in Production mode.
- When enabled:
 - The administration server monitors its `autodeploy` folder for new, updated, or removed applications
 - Applications are targeted to the administration server only
 - Developers can quickly test or experiment with an application



Auto-Deployment is a method for quickly deploying an application to a stand-alone server (Administration Server) for evaluation or testing. It is recommended that this method be used only in a single-server development environment. You can run a WebLogic Server domain in two different modes: development and production. You can use the Auto-Deployment feature only in Development mode.

If auto-deployment is enabled, when an application is copied into the `\autodeploy` directory of the Administration Server, the Administration Server detects the presence of the new application and deploys it automatically (if the Administration Server is running). If WebLogic Server is not running when you copy the application to the `\autodeploy` directory, the application is deployed the next time the WebLogic Server Administration Server is started. Auto-deployment deploys only to the Administration Server.

A deployment unit that was auto-deployed can be dynamically redeployed while the server is running. To dynamically redeploy, copy the new version of the archive file over the existing file in the `/autodeploy` directory. When an application has been auto-deployed in exploded archive format, the Administration Server periodically looks for a file named `REDEPLOY` in the exploded application directory. If the timestamp on this file changes, the Administration Server redeploys the exploded directory.

- WebLogic's *FastSwap* feature is:
 - Enabled using WebLogic deployment descriptors
 - Available only if the domain is **not** running in Production mode
 - Applicable only to Web applications that are **not** archived
- When enabled:
 - WebLogic automatically reloads modified Java class files within applications
 - Developers can perform iterative development without an explicit redeployment

Excerpt from weblogic.xml:

```
<fast-swap>true</fast-swap>
```

With FastSwap, Java classes are redefined in-place without reloading the classloader, thereby having the decided advantage of fast turnaround times. This means that you do not have to wait for an application to redeploy and then navigate back to wherever you were in the Web page flow. Instead, you can make your changes, compile, and then see the effects immediately.

FastSwap is only supported when WLS is running in Development mode. It is automatically disabled in Production mode. Only changes to class files in exploded directories are supported. Modifications to class files in archived applications, as well as archived JAR files appearing in the application's classpath are not supported. Within an exploded Web application, modifications to Java classes are only supported in the `WEB-INF/classes` directory.

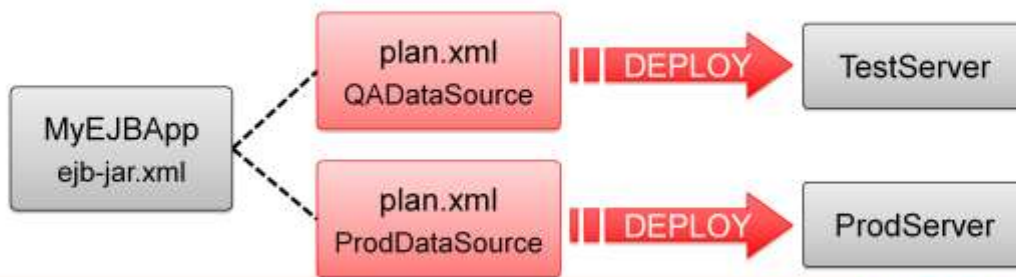
To enable FastSwap in your application, add the `<fast-swap>` element to the `weblogic-application.xml` file for an enterprise application, or to the `weblogic.xml` file for a stand-alone Web application.

For headless applications (that is, applications not fronted by a Web application), class redefinition can be explicitly initiated using WebLogic's JMX interface. For convenience, an Ant task that uses this JMX interface is also available—`com.bea.wls.redef.ant.FastSwapTask`.

Deployment Plans

A Java EE *Deployment Plan*:

- Is an XML file associated with an application
- Resides outside an application archive
- Sets or overrides values in Java EE deployment descriptors
- Allows a single application to be easily customized to multiple deployment environments



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

A deployment plan is an XML document used to define an application's deployment configuration for a specific WebLogic Server environment, such as development, test, or production. A deployment plan resides outside of an application's archive file and contains deployment properties that override an application's existing Java EE and WebLogic Server deployment descriptors. Use deployment plans to easily change an application's WebLogic Server configuration for a specific environment without modifying existing deployment descriptors. Multiple deployment plans can be used to reconfigure a single application for deployment to multiple, differing WebLogic Server environments.

Any external resources required by the application are subject to change when the application is deployed to a different environment. For example, the JNDI names of data sources used in your development environment can be different from those used in testing or production. Exposing those JNDI names as variables makes it easy for deployers to use available resources or create required resources when deploying the application.

Certain tuning parameters that are acceptable in a development environment are unacceptable in a production environment. For example, it may suffice to accept default or minimal values for EJB caching on a development machine, whereas a production cluster would need higher levels of caching to maintain acceptable performance.

Create a New Deployment Plan

WLS includes tools to accelerate deployment plan creation.

The administration console:

- Generates a skeleton `plan.xml` if a `plan` folder is detected with a newly deployed application
- Updates `plan.xml` when you use the console to modify deployment descriptor settings

The `weblogic.PlanGenerator` Java class can also generate a skeleton `plan.xml` for an existing application.

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

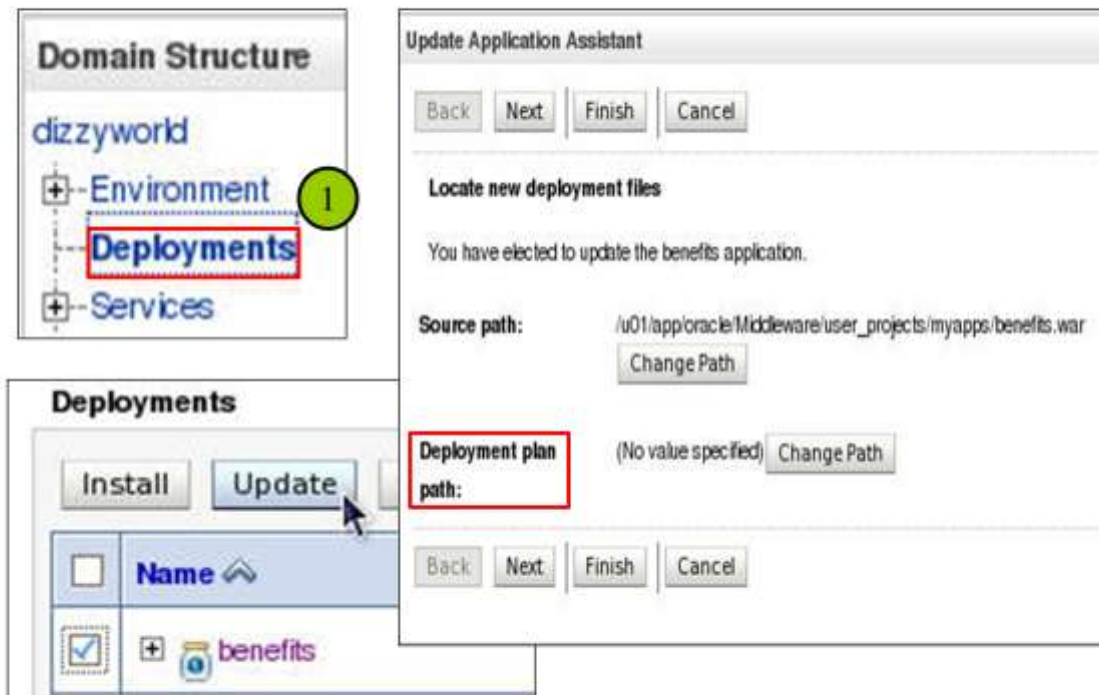
To create a deployment plan for a deployed application that does not already have a deployment plan, make a configuration change to the deployed application using the administration console. When you make a persisted configuration change to a deployed application that does not have an existing deployment plan, the console automatically creates a deployment plan for you and prompts you for the location in which to save it.

`weblogic.PlanGenerator` is a Java-based deployment configuration tool intended for developers who want to export portions of a WebLogic Server deployment configuration into a deployment plan. This utility can generate a brand new plan, or can append to an existing one.

By default, `weblogic.PlanGenerator` writes an application's deployment plan to a file named `plan.xml` in the application's root directory. The syntax for invoking `weblogic.PlanGenerator` is the following:

```
java weblogic.PlanGenerator [options] [application]
```

Use an Existing Deployment Plan



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

You can use the administration console to specify a deployment plan for your application.

1. In the left pane, click Deployments.
2. In the right pane, select the check box next to the application for which you want to specify a deployment plan. Click Update.
3. Click Change Path next to "Deployment plan path" to browse to the desired deployment plan. Click Next, and then click Finish.

Application Availability

- By default, when an application is redeployed:
 - It is unavailable to clients for a brief time
 - Existing clients lose any conversational state
- Some types of applications require availability 24 hours a day, 7 days a week.
- Third-party proxy solutions are possible, but they require multiple servers.



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In a production environment, deployed applications frequently require 24×7 availability to provide uninterrupted services to customers and internal clients. In these cases, scheduling maintenance down time to replace applications is not desirable.

By default, WebLogic uses an in-place redeployment scheme. In-place redeployment immediately replaces a running application's deployment files with updated deployment files. In contrast to production redeployment, in-place redeployment of an application or stand-alone J2EE module does not guarantee uninterrupted service to the application's clients. This is because WebLogic Server immediately removes the running classloader for the application and replaces it with a new classloader that loads the updated application class files.

Production Redeployment:

- Allows two versions of a single Web application or module to run simultaneously
- Requires you to include unique version information either:
 - Within the application's `META-INF/MANIFEST.MF` file
 - As part of the deployment process

When a new version is redeployed, WLS automatically:

- Routes existing clients to the prior (retired) version
- Routes new clients to the new version
- Undeploys the prior version when all existing clients finish their work or their conversations timeout

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

WebLogic's production redeployment strategy involves deploying a new version of an updated application alongside an older version of the same application. WebLogic Server automatically manages client connections so that only new client requests are directed to the new version. Clients already connected to the application during the redeployment continue to use the older version of the application until they complete their work, at which point WebLogic Server automatically retires the older application. Production redeployment is currently not supported for stand-alone EJB applications.

When you redeploy a new version of an application, WebLogic Server treats the newly deployed application version as the active version, and begins retiring the older version. During the retirement period, WebLogic Server automatically tracks the application's HTTP sessions and in-progress transactions. WebLogic Server tracks each HTTP session until the session completes or has timed out. In-progress transactions are tracked until the transaction completes, rolls-back, or reaches the transaction timeout period.

To assign a version identifier to an application, Oracle recommends that you store a unique version string directly in the `MANIFEST.MF` file of the EAR or WAR being deployed. Alternatively, specify a version in the administration console by using the `appversion` argument of `weblogic.Deployer`.

Monitor an Application

The available monitoring features vary by application type.

The screenshot shows the Oracle Administration Console interface. On the left, the 'Domain Structure' pane shows a tree view for 'dizzyworld' with 'Environment' and 'Services' expanded. 'Deployments' is highlighted with a red box and a green circle labeled '1'. On the right, the 'Settings for benefits' pane shows the 'Monitoring' tab selected. Below it, the 'Web Applications' tab is selected, and a table titled 'Web Applications' is displayed. The table has columns: State, Sessions, Sessions High, Total Sessions, and Servlets. The data row shows: Active, 3, 3, 4, and 5. A green circle labeled '2' is next to the 'Web Applications' tab.

| State | Sessions | Sessions High | Total Sessions | Servlets |
|--------|----------|---------------|----------------|----------|
| Active | 3 | 3 | 4 | 5 |

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In the left pane of the Administration Console, select Deployments. In the right pane, click the application you want to monitor.

Click the Monitoring tab. The available second level tabs will vary depending on the type of the application. For example, a Web application or module has tabs named Web Application, Servlets, Sessions, and Workload.

For Web applications, the available columns include:

Context Root: Returns the context root (context path) for the Web application.

Servlets: The number of Java Servlets deployed within this application, including internal WebLogic Servlets. If desired, the Servlets second level tab displays statistics on a per-servlet basis.

Sessions: Provides a count of the current number of open sessions in this module.

Sessions High: Provides the highest number of active sessions ever managed by this application. The count starts at zero each time the application is activated.

Total Sessions: Provides a count of the total number of sessions opened since the application was deployed.

Lesson Review



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Now let's see how well you understand WebLogic's core deployment lifecycle, features and tools. Answer the following series of simple questions.



Place holder for quiz

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In this course, you learned how to:

- Describe the types of components within a typical Java EE application
- Discuss the differences between an application and a library in WLS
- Create and deploy a simple JDBC data source
- Create and deploy simple JMS destinations
- Perform common deployment tasks using the console and command line
- Define the terms “deployment plan” and “production redeployment”

Let's do a quick recap of some of the topics we covered in this self-study course. Recall that we learned how to:

Describe the types of components within a typical Java EE application

Discuss the differences between an application and a library in WebLogic Server

Create and deploy a simple JDBC data source

Create and deploy simple JMS destinations

Perform common deployment tasks using the console and command line

Define the terms “deployment plan” and “production redeployment” as they relate to WebLogic Server

How Can I Learn More?

Self-Study Courses


Instructor-led Courses

Other Resources

More Information

There are a variety of channels from which you can learn more about Enterprise Manager Cloud Control. Click on a tab on the left for more information about just a few of the possibilities.


We hope you found this sample self-study course informative and useful.



PROPERTIES
Allow user to leave interaction:
Show 'Next Slide' Button:
Completion Button Label:

[Anytime](#)
[Show upon completion](#)
[Next Slide](#)

Properties...

 Edit in Engage

Oracle offers a variety of additional channels from which you can learn more about other Oracle WebLogic Server or about any Oracle products. We at Oracle Education know your time is valuable and limited, so we thank you for participating in this self-paced training. We hope this course has met your expectations and learning objectives, and wish you the best of luck in all of your endeavors. Thanks again.