Oracle® Fusion Middleware

Deploying Applications to Oracle WebLogic Server

14c (14.1.2.0.0)
F72063-02
March 2025

Oracle Fusion Middleware Deploying Applications to Oracle WebLogic Server, 14c (14.1.2.0.0)

F72063-02

Contents

2 Preparing Applications and Modules for Deployment

3 Configuring Applications for Production Deployment

A weblogic.Deployer Command-Line Reference

B weblogic.PlanGenerator Command-Line Reference

Preface

This document describes deploying Jakarta EE applications or application modules to Oracle WebLogic Server 14c instances or clusters.

Audience

This document is a resource for:

- Administrators who want to deploy Jakarta EE applications or application modules to WebLogic Server instances or clusters. It is assumed that you are working in a production environment, which is generally characterized by multiple WebLogic Server instances or clusters running on multiple machines. It is also assumed that you have one or more application module archive files that have been tested and are ready to deploy on a production server.
- Developers who may need to deploy an application in a development environment, package an application for delivery to an administrator or deployer, or create and export the configuration of an application for deployment to a testing, staging, or production environment

Documentation Accessibility

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

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

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

Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

For additional information about deploying applications and modules to WebLogic Server, see these documents:

- *Deploying Applications with the WebLogic Deployment API* describes the WebLogic Server deployment API, which implements and extends the Jakarta EE specification. All WebLogic Server deployment tools use this API.
- *Developing Applications for Oracle WebLogic Server* describes how to deploy applications during development using the `wldeploy` Ant task, and provides information about the WebLogic Server deployment descriptor for enterprise applications.
- The WebLogic Server Jakarta EE programming guides describe the Jakarta EE and WebLogic Server deployment descriptors used with each Jakarta EE application and module:
  - *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*
  - *Developing Enterprise JavaBeans, Version 3.2, for Oracle WebLogic Server*
  - *Developing Resource Adapters for Oracle WebLogic Server*
  - *Developing JAX-WS Web Services for Oracle WebLogic Server*
  - *Deploying Applications with the WebLogic Deployment API*
- *Developing JDBC Applications for Oracle WebLogic Server* describes the XML deployment descriptors for JDBC application modules.
- *Developing JMS Applications for Oracle WebLogic Server* describes the XML deployment descriptors for JMS application modules.
- New and Changed Features in This Release

New and Changed Features in This Release

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

Conventions

The following text conventions are used in this document:

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

1

Understanding WebLogic Server Deployment

Learn about WebLogic Server application deployment features.
This chapter includes the following sections:

- Overview of the Deployment Process
  The term *application deployment* refers to the process of making an application or module available for processing client requests in a WebLogic Server domain. Application deployment involves multiple tasks.
- Deployment Terminology
  These WebLogic Server deployment terms are used throughout this document.
- Standards Compatibility
  WebLogic Server implements the Jakarta EE specification which includes a deployment specification, JSR-88, that describes a standard API used by deployment tools and application server providers to configure and deploy applications to an application server.
- Jakarta EE Deployment Implementation
  WebLogic Server implements both the JSR-88 Service Provider Interface (SPI) plug-in and model plug-in to comply with the Jakarta EE deployment specification.
- WebLogic Server Deployment Features
  WebLogic Server supports several advanced deployment features to help you reliably deploy and manage applications in a production environment.
- Supported Deployment Units
  A *deployment unit* refers to a Jakarta EE application (an enterprise application or Web application) or a standalone Jakarta EE module (such as an EJB or resource adapter) that has been organized according to the Jakarta EE specification and can be deployed to WebLogic Server.
- Deployment Tools
  WebLogic Server provides several tools to help you configure and deploy applications.

Overview of the Deployment Process

The term *application deployment* refers to the process of making an application or module available for processing client requests in a WebLogic Server domain. Application deployment involves multiple tasks.

- Preparing Applications and Modules for Deployment
- Configuring Applications for Production Deployment
- Exporting an Application for Deployment to New Environments
- Deploying Applications and Modules with weblogic.Deployer
- Auto-Deploying Applications in Development Domains
- Redeploying Applications in a Production Environment
- Managing Deployed Applications

Deployment Terminology

These WebLogic Server deployment terms are used throughout this document.

- *application*—One or more software programs, used collectively by an end user to perform computing tasks.
- *application installation directory*—A WebLogic Server directory structure designed to help organize deployment files and generated deployment configuration artifacts for an application or module. Also referred to as an *application root directory*.
- *application module*—An XML document used to configure JMS or JDBC resources. An application module can be one of the following types:
  - *standalone*— Resources are bound to the global JNDI tree.
  - *application-scoped*—Bundled as part of an enterprise application and scoped within the application itself.
- *application version*—A string value that identifies the version of a deployed application. Compatible applications that use version strings can use the WebLogic Server production redeployment strategy.
- *deployment configuration*—The process of defining the deployment descriptor values required to deploy an application to a particular WebLogic Server domain. The deployment configuration for an application or module is stored in three types of XML document: Jakarta EE deployment descriptors, WebLogic Server descriptors, and WebLogic Server deployment plans.
- *deployment descriptor*—An XML document used to define the Jakarta EE behavior or WebLogic Server configuration of an application or module at deployment time.
- *deployment plan*—An XML document used to define an application's WebLogic Server 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 which override an application's existing 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.
- *distribution*—The process by which WebLogic Server copies deployment source files to target servers for deployment.
- *production redeployment*—A WebLogic Server redeployment strategy that deploys a new version of a production application alongside an older version, while automatically managing HTTP connections to ensure uninterrupted client access.
- *staging mode*—The method WebLogic Server uses to make deployment files available to target servers in a domain. Staging modes determine whether or not files are distributed (copied) to target servers before deployment.

Standards Compatibility

WebLogic Server implements the Jakarta EE specification which includes a deployment specification, JSR-88, that describes a standard API used by deployment tools and application server providers to configure and deploy applications to an application server.

> **Note:**
>
> The Jakarta EE Deployment API specification (JSR-88) provides the same functionality as the Jakarta Deployment 1.7 specification. All references in this document to the Jakarta EE Deployment API specification (JSR-88) can be interpreted as references to the Jakarta Deployment 1.7 specification.

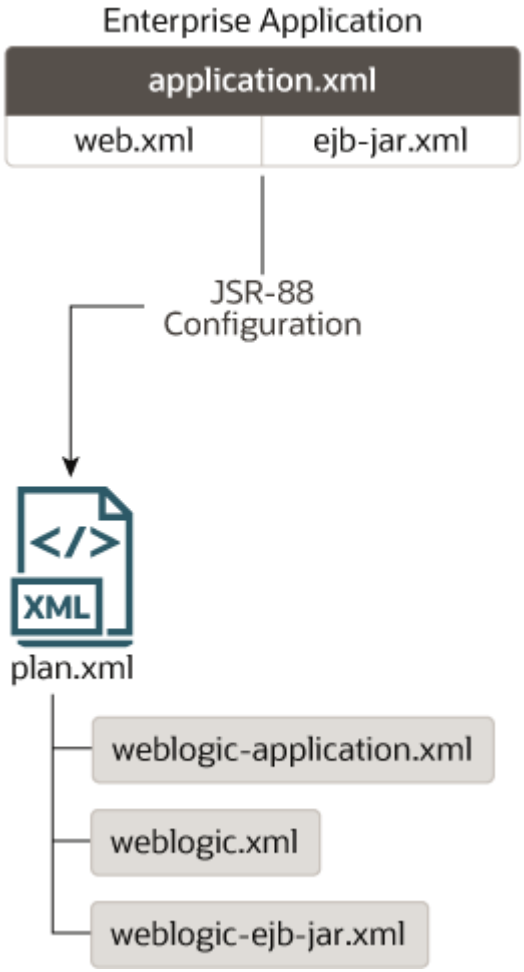
WebLogic Server implements both the JSR-88 Service Provider Interface (SPI) plug-in and model plug-in to comply with the Jakarta EE deployment specification. You can use a basic Jakarta EE deployment API deployment tool with the WebLogic Server plug-ins (without using WebLogic Server extensions to the API) to configure, deploy, and redeploy Jakarta EE applications and modules to WebLogic Server. See WebLogic Server Compatibility in *Understanding Oracle WebLogic Server*.

Jakarta EE Deployment Implementation

WebLogic Server implements both the JSR-88 Service Provider Interface (SPI) plug-in and model plug-in to comply with the Jakarta EE deployment specification.

You can use a basic Jakarta EE deployment API deployment tool with the WebLogic Server plug-ins (without using WebLogic Server extensions to the API) to configure, deploy, and redeploy Jakarta EE applications and modules to WebLogic Server. The WebLogic Server configuration generated by a Jakarta EE deployment API configuration process is stored in a deployment plan and one or more generated WebLogic Server deployment descriptor files, as shown in Figure 1-1. WebLogic Server deployment descriptors are generated as needed to store WebLogic Server configuration data.

Figure 1-1 Configuring Applications with the Jakarta EE Deployment API



The WebLogic Server deployment plan generated by a Jakarta EE deployment API deployment tool identifies the WebLogic Server deployment descriptors that were generated for the application during the configuration session.

Although the Jakarta EE deployment API provides a simple, standardized way to configure applications and modules for use with a Jakarta EE-compliant application server, the specification does not address many deployment features that were available in previous WebLogic Server releases. For this reason, WebLogic Server provides important extensions to the Jakarta EE deployment API specification to support capabilities described in WebLogic Server Deployment Features.

WebLogic Server Deployment Features

WebLogic Server supports several advanced deployment features to help you reliably deploy and manage applications in a production environment.

- Additional Deployment Configuration Properties
- Exporting Applications for Deployment to Multiple Environments
- Administration Mode for Isolating Production Applications
- Deployable JDBC, JMS, and WLDF Application Modules

- Module-Level Deployment and Redeployment for Enterprise Applications
- Safe Redeployment for Production Applications
- Security Roles Required for Deployment

Additional Deployment Configuration Properties

Whereas the Jakarta EE deployment API deployment specification enables you to generate vendor-specific descriptor values necessary for deploying an application, WebLogic Server extensions to Jakarta EE deployment API allow you to configure many additional deployment properties, including:

- The names of external resources required for the application to operate
- The declared names of services provided in a deployed application (JNDI names), which other applications may reference for their own use
- Tuning properties that control the performance and behavior of the application on WebLogic Server

You can store these deployment properties in WebLogic Server deployment plans.

Exporting Applications for Deployment to Multiple Environments

The basic Jakarta EE deployment API configuration process provides a simple way for standardized deployment tools to deploy Jakarta EE applications on multiple application server products. However, it does not help in the process of migrating an application's configuration from one environment to another within an organization. The WebLogic Server deployment API extends the Jakarta EE deployment API to provide support for exporting an application's configuration for deployment to multiple WebLogic Server environments, such as testing, staging, and production domains. See Exporting an Application for Deployment to New Environments.

Administration Mode for Isolating Production Applications

Distributing an application copies deployment files to target servers and places the application in a prepared state. You can then start the application in administration mode, which restricts access to the application to a configured administration channel so you can perform final testing without opening the application to external client connections or disrupting connected clients. You can start an application in administration mode with the `-adminmode` option as described in Starting a Distributed Application in Administration Mode. See Distributing Applications to a Production Environment and Distributing a New Version of a Production Application.

After performing final testing, you can either undeploy the application to make further changes, or start the application in Production mode to make it generally available to clients.

See Distributing an Application.

Deployable JDBC, JMS, and WLDF Application Modules

JDBC, JMS, and WLDF resources can be stored as application modules, which can be deployed standalone to multiple servers or clusters or included within an enterprise application as application-scoped resources. Standalone JDBC, JMS, and WLDF application modules make it easy to replicate resources in multiple WebLogic Server domains. Application-scoped resource modules make it possible to include all of an application's required resources within the application module itself, for maximum portability to multiple environments. See *Developing*

Applications for Oracle WebLogic Server for more information about using application-scoped resources. See Deploying JDBC, JMS, and WLDF Application Modules to deploy standalone or application-scoped resources to WebLogic Server.

Module-Level Deployment and Redeployment for Enterprise Applications

WebLogic Server enables you to target individual modules of an enterprise application to different server targets, or to deploy only a subset of available modules in an enterprise application. This provides flexible packaging options, allowing you to bundle a group of related modules together in an enterprise application, but deploy only selected modules to individual servers in a domain.

Safe Redeployment for Production Applications

WebLogic Server enables you to safely update and redeploy a new version of a production application without affecting current HTTP clients to the application. Production redeployment helps you roll out bug fixes or new functionality without application downtime, and without creating redundant servers in order to roll out the changes. See Redeploying Applications in a Production Environment.

Security Roles Required for Deployment

The built-in security roles for "Admin" and "Deployer" users allow you to perform deployment tasks using the WebLogic Remote Console. The "AppTester" security role allows you to test versions of applications that are deployed to administration mode. When deploying across WebLogic domains, the "CrossDomainConnector" role allows you to make inter-domain calls from foreign domains. For a complete listing of all security roles, see Default Global Roles in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.

Supported Deployment Units

A *deployment unit* refers to a Jakarta EE application (an enterprise application or Web application) or a standalone Jakarta EE module (such as an EJB or resource adapter) that has been organized according to the Jakarta EE specification and can be deployed to WebLogic Server.

For each type of deployment unit, the Jakarta EE specification defines both the required files and their location in the directory structure of the application or module. Deployment units may include Java classes for EJBs and servlets, resource adapters, Web pages and supporting files, XML-formatted deployment descriptors, and even other modules.

Jakarta EE does not specify *how* a deployment unit is deployed on the target server—only how standard applications and modules are organized. WebLogic Server supports the following types of deployment units:

- Enterprise Application
- Web Application
- Jakarta Enterprise Beans
- Resource Adapter
- Web Service
- Jakarta EE Library
- Optional Package

- JDBC, JMS, and WLDF Modules
- DBClientData Modules
- Client Application Archive

Enterprise Application

An enterprise application consists of one or more of the following Jakarta EE applications or modules:

- Web applications
- Jakarta Enterprise Beans (EJB) modules
- Resource adapter modules

An enterprise application is packaged as an archive file with an `.ear` extension, but is generally deployed as an exploded EAR directory. An exploded EAR directory contains all of the JAR, WAR, and RAR modules (also in exploded format) for an application as well as the XML descriptor files for the enterprise application and its bundled applications and modules. See *Developing Applications for Oracle WebLogic Server*.

Web Application

A Web application always includes the following files:

- A servlet or JSP page, along with any helper classes.
- A `web.xml` deployment descriptor, a Jakarta EE standard XML document that configures the contents of a WAR file.

Web applications may also contain JSP tag libraries, static `.html` and image files, supporting classes and `.jar` files, and a `weblogic.xml` deployment descriptor, which configures WebLogic Server-specific elements for Web applications. See *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

Jakarta Enterprise Beans

Jakarta Enterprise Beans (EJBs) are reusable Java components that implement business logic and enable you to develop component-based distributed business applications. EJB modules are packaged as archive files having a `.jar` extension, but are generally deployed as exploded archive directories. The archive file or exploded archive directory for an EJB contains the compiled EJB classes, optional generated classes, and XML deployment descriptors for the EJB. See *Developing Enterprise JavaBeans, Version 2.1, for Oracle WebLogic Server* for more information on the different types of EJBs.

Resource Adapter

A resource adapter (also referred to as a connector) adds Enterprise Information System (EIS) integration to the Jakarta EE platform. Connectors provide a system-level software driver that WebLogic Server can use to connect to an EIS. Connectors contain both the Java classes, and if necessary, the native components required to interact with the EIS. See *Developing Resource Adapters for Oracle WebLogic Server*.

Web Service

A Web service is a set of functions packaged into a single entity that is available to other systems on a network, and can be shared by and used as a component of distributed Web-based applications. Web services commonly interface with existing back-end applications, such as customer relationship management systems, order-processing systems, and so on.

A Web service module may include either Java classes or EJBs that implement the Web service. Web services are packaged either as Web application archives (WARs) or EJB modules (JARs) depending on the implementation; typically the WAR or EJB JAR file is then packaged in an enterprise application. See *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Jakarta EE Library

A Jakarta EE library is a standalone Jakarta EE module, or multiple Jakarta EE modules packaged in an enterprise application (EAR), that is registered with the Jakarta EE application container as a shared library at deployment time. After a Jakarta EE library has been registered, you can deploy enterprise applications that reference the library in their `weblogic-application.xml` deployment descriptors. Each referencing application receives a copy of the shared Jakarta EE library module(s) on deployment, and can use those modules as if they were packaged as part of the application itself. Jakarta EE library support provides an easy way to share one or more Jakarta EE modules among multiple enterprise applications without physically adding the shared modules to each dependent application.

The deployment files of a shared library resemble either a standard enterprise application or Jakarta EE module, as discussed in this section. Shared libraries differ from standard EARs and modules only by the contents of their `MANIFEST.MF` files. Creating Shared Jakarta EE Libraries and Optional Packages in *Developing Applications for Oracle WebLogic Server* describes how to assemble and configure Jakarta EE libraries, and how to configure enterprise applications that utilize Jakarta EE libraries.

Deploying Shared Jakarta EE Libraries and Dependent Applications describes how to deploy Jakarta EE libraries and enterprise applications that reference Jakarta EE libraries.

Optional Package

Optional packages provide similar functionality to Jakarta EE libraries, allowing you to easily share a single JAR file among multiple applications. However, optional packages function as individual Jakarta EE modules (standalone or within an enterprise application), rather than as an enterprise application. For example, third-party Web Application Framework classes needed by multiple Web applications can be packaged and deployed in a single JAR file, and referenced by multiple Web application modules in the domain.

Optional packages are delivered as basic JAR files that have no deployment descriptors. You simply designate the JAR as an optional package at deployment time, and WebLogic Server registers the file with the target servers you select. After the optional package has been registered, you can then deploy Jakarta EE modules and applications that reference the optional package in their `MANIFEST.MF` files. Creating Shared Jakarta EE Libraries and Optional Packages in *Developing Applications for Oracle WebLogic Server* describes how to assemble and configure optional packages, and how to configure Jakarta EE modules that utilize optional packages.

Deploying Shared Jakarta EE Libraries and Dependent Applications describes how to deploy optional packages and Jakarta EE modules that reference optional packages.

JDBC, JMS, and WLDF Modules

A JMS, JDBC, or WebLogic Diagnostic Framework (WLDF) application module can be deployed as a standalone resource, in which case the resource is available in the domain targeted during deployment, or as part of an enterprise application. An application module deployed as part of an enterprise application is available only to the enclosing application (an *application-scoped resource*). Using application-scoped resources ensures that an application always has access to required resources, and simplifies the process of deploying the application into new environments.

In contrast to system modules, application modules are owned by the developer who created and packaged the module, rather than the administrator who deploys the module. This means that the administrator has more limited control over JDBC, JMS, and WLDF application modules. When deploying an application module, an administrator can change resource properties that were specified in the module, but cannot add or delete resources.

System modules are created by the administrator using the WebLogic Remote Console, and can be changed or deleted as necessary by the administrator. Similarly, standalone application modules created by the administrator can be used to recreate global resources in multiple WebLogic Server environments simply by deploying the modules into new domains.

For more information on how to deploy and use JDBC, JMS, and WLDF modules, see:

- Deploying JDBC, JMS, and WLDF Application Modules
- Deploying Oracle Wallet Files for JDBC Modules
- *Administering JMS Resources for Oracle WebLogic Server*
- *Administering JDBC Data Sources for Oracle WebLogic Server*
- *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*

DBClientData Modules

DBClientData modules are standalone deployment modules that, when present, are always used with JDBC modules. They may contain `tnsnames.ora` files, Oracle wallet files, server trust keystores, client identity keystores, basically all the database client connection data used by a data source, but are independent of the configuration of the data source instances that are going to use those data. For more information, see Deploying Oracle Wallet Files for JDBC Modules.

DBClientData modules make it easy to deploy, manage, and move database client data. For more information, see Use DBClientData Modules for Portability in *Administering JDBC Data Sources for Oracle WebLogic Server*.

Client Application Archive

The Jakarta EE specification enables you to include a client application archive file (`.jar` file) as a module within an enterprise application (`.ear` file). A Jakarta EE client application module contains the Java classes that execute in the client JVM (Java Virtual Machine) and deployment descriptors that describe EJBs (Enterprise JavaBeans) and other WebLogic Server resources used by the client. This enables both the server-side and client-side components to be distributed as a single unit. You define client modules in an EAR using the Jakarta EE standard `application-client.xml` deployment descriptor and WebLogic Server `weblogic-appclient.xml` descriptor.

Deployment Tools

WebLogic Server provides several tools to help you configure and deploy applications.

- weblogic.Deployer
- WebLogic Remote Console and Fusion Middleware Control
- WLST
- Deployment Tools for Developers

weblogic.Deployer

`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 WebLogic Remote Console.

WebLogic Remote Console and Fusion Middleware Control

The WebLogic Remote Console and Fusion Middleware Control guide you through the deployment process. They also provide controls for changing and monitoring the deployment status, and changing selected deployment descriptor values while the deployment unit is up and running.

Use the WebLogic Remote Console or Fusion Middleware Control when you need to perform basic deployment functions interactively and you have access to a supported browser.

WLST

The WebLogic Scripting Tool (WLST) is a command-line interface that you can use to automate domain configuration tasks, including application deployment configuration and deployment operations. See *Understanding the WebLogic Scripting Tool* for more information.

Deployment Tools for Developers

WebLogic Server provides several tools for deploying applications and standalone modules:

- `wldeploy` is an Ant task 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.
- `weblogic-maven-plugin` is a Maven plug-in for WebLogic Server that you can use to perform deployment operations similar to those supported by `weblogic.Deployer`. The plug-in lets you deploy, redeploy, update, and such, applications built using Maven to WebLogic Server from within the Maven environment.
- `weblogic.PlanGenerator` is a command-line tools that enables developers to export an application's configuration for deployment to multiple WebLogic Server environments.
- The deployment API allows you to perform deployment tasks programmatically using Java classes.
- The `autodeploy` domain directory allows you to deploy an application quickly for evaluation or testing in a development environment.

2

Preparing Applications and Modules for Deployment

Learn how to prepare applications and modules for deployment to WebLogic Server.
This chapter includes the following sections:

- Packaging Files for Deployment
  WebLogic Server supports deployments that are packaged either as archive files using the `jar` utility or Ant's `jar` tool, or as exploded archive directories.
- Understanding Default Deployment Names
  When you first deploy an application or standalone module to one or more WebLogic Server instances, you specify a deployment name to describe collectively the deployment files, target servers, and other configuration options you selected. You can later redeploy or stop the deployment unit on all target servers by simply using the deployment name. The deployment name saves you the trouble of re-identifying the deployment files and target servers when you want to work with the deployment unit across servers in a domain.
- Understanding Application Naming Requirements
  In order to successfully deploy an application to WebLogic Server, the application name must be valid.
- Understanding Deployment Version Strings
  In addition to a deployment name, an application or module can also have an associated version string. The version string distinguishes the initial deployment of the application from subsequent redeployed versions.
- Creating an Application Installation Directory
  The application installation directory separates generated configuration files from the core application files, so that configuration files can be easily changed or replaced without disturbing the application itself. The directory structure also helps you to organize and maintain multiple versions of the same application deployment files.
- Using FastSwap Deployment to Minimize Redeployment
  Today's Web application developers expect to make changes to a deployed application and see those changes immediately by refreshing the browser.
- Best Practices for Preparing Deployment Files
  Follow Oracle-recommended best practices when preparing applications and modules for deployment.

Packaging Files for Deployment

WebLogic Server supports deployments that are packaged either as archive files using the `jar` utility or Ant's `jar` tool, or as exploded archive directories.

> **Note:**
>
> In general, using archived files is more efficient when deploying applications to Managed Servers. However, it makes updating the application, such as updating Web content, more difficult as it requires a redeployment of the application.

Using Archived Files

An archive file is a single file that contains all of an application's or module's classes, static files, directories, and deployment descriptor files. In most production environments, the applications an administrator receives for deployment are stored as archive files.

Deployment units that are packaged using the `jar` utility have a specific file extension depending on the type:

- EJBs and client archives are packaged as `.jar` files.
- Web applications are packaged as `.war` files.
- Resource adapters are packaged as `.rar` files.
- Enterprise applications are packaged as `.ear` files, and can contain other Jakarta EE modules such as Web archives (`.war` files), EJB archives (`.jar` files), connector archives (`.rar` files) and client archives (`.jar` files).
- Web services can be packaged either as `.war` files or as `.jar` files, depending on whether they are implemented using Java classes or EJBs. Typically, the `.war` or `.jar` files are then packaged in an enterprise application `.ear` file.
- Jakarta EE libraries are packaged either as an enterprise application (`.ear` file) or as a standard Jakarta EE module.
- Client applications and optional packages are packaged as `.jar` files.

In addition to an archive file, you may also receive a deployment plan, which is a separate file that configures the application for a specific environment. Configuring Applications for Production Deployment describes deployment plans in more detail.

Using Exploded Archive Directories

An exploded archive directory contains the same files and directories as a JAR archive. If you choose to use an exploded archive directory, you may be required to manually unpack a previously-archived deployment. However, the files and directories reside directly in your file system and are not packaged into a single archive file with the `jar` utility.

You may choose to deploy from an exploded archive directory under the following circumstances:

- You want to perform partial updates of an enterprise application after deployment. Deploying enterprise applications as an exploded archive directory makes it easier to update individual modules of the application without having to re-create the archive file.

- You are deploying a Web application or enterprise application that contains static files that you will periodically update. In this case, it is more convenient to deploy the application as an exploded directory, because you can update and refresh the static files without re-creating the archive.
- You are deploying a Web application that performs direct file system I/O through the application context (for example, a Web application that tries to dynamically edit or update parts of the Web application itself). In this case, the modules that perform the I/O operations should have a physical file system directory in which to work; you cannot obtain a file when the application is deployed as an archive, as per the specification.
- Jakarta EE Rules for Deploying Exploded EAR Directories without Deployment Descriptors

Jakarta EE Rules for Deploying Exploded EAR Directories without Deployment Descriptors

The Jakarta EE specification recommends that archived EARs (Enterprise Application Archives) can be deployed to a Jakarta EE-compliant server without any deployment descriptors. To achieve this, all containers assume reasonable defaults *or* use annotated classes. In addition to supporting this mandate, WebLogic Server also allows deploying exploded EAR directories without deployment descriptors.

Because this applies to directories, certain rules are used to identify EARs and their nested modules. Otherwise, the WebLogic Remote Console or deployment tools will not treat the directories as valid exploded Jakarta EE directories.

- For an exploded archived Web application, in the absence of `WEB-INF/web.xml` descriptor, the name of the directory should have a `.war` suffix.
- For an exploded archived enterprise application without a `META-INF/application.xml` descriptor, the directory should have an `.ear` suffix. Within the application, the directory of exploded Web module should have a `.war` suffix. Similarly, the exploded EJB module should have a `.jar` suffix and the exploded RAR module should have a `.rar` suffix.
- If an exploded enterprise application contains no `META-INF/application.xml` descriptor, the order in which modules are deployed is undefined and is dependent on the underlying `File.listFiles()` method order. To ensure a specific order in which modules are deployed, you must add an `application.xml` descriptor and list the modules in the desired order.

Creating an Exploded Archive Directory from an Archive File

If you have an archive file that you want to deploy as an exploded archive directory, use the `jar` utility to unpack the archive file in a dedicated directory. For example:

```
mkdir /myapp
cd /myapp
jar xvf /dist/myapp.ear
```

If you are unpacking an archive file that contains other module archive files (for example, an enterprise application or Web Service that includes JAR or WAR files) and you want to perform partial updates of those modules, you must expand the embedded archive files as well. Make sure that you unpack each module into a subdirectory having the same name as the archive file. For example, unpack a module named `myejb.jar` into a `/myejb.jar` subdirectory of the exploded enterprise application directory.

> **Note:**
>
> If you want to use different subdirectory names for the archived modules in an exploded EAR file, you must modify any references to those modules in the application itself. For example, you must update the URI values specified in `application.xml` and `CLASSPATH` entries in the `manifest.mf` file.

Understanding Default Deployment Names

When you first deploy an application or standalone module to one or more WebLogic Server instances, you specify a deployment name to describe collectively the deployment files, target servers, and other configuration options you selected. You can later redeploy or stop the deployment unit on all target servers by simply using the deployment name. The deployment name saves you the trouble of re-identifying the deployment files and target servers when you want to work with the deployment unit across servers in a domain.

If you do not specify a deployment name at deployment time, the deployment tool selects a default name based on the deployment source file(s). For archive files, `weblogic.Deployer` uses the name of the archive file without the file extension. For example, the file `myear.ear` has a default deployment name of `myear`. For an exploded archive directory, `weblogic.Deployer` uses the name of the top-level directory you deploy. Auto-deployed applications and standalone modules get a computed name exactly as described for other applications; they will neither have the file extension nor be pre-pended with a string that starts with an underscore.

For Jakarta EE libraries and optional packages, `weblogic.Deployer` uses the name specified in the library's manifest file. If no name was specified in the library's manifest file, you can specify one with the `-name` option.

As of WebLogic Server 12.1.1, in compliance with Java EE 6 specifications, there is an additional way of specifying an application name, the `<application-name>` element in `application.xml`. For standalone modules, you can specify the `<module-name>` element in the respective deployment descriptor. Application names provided in deployment descriptors (`application-name` in `application.xml` or `module-name` in `web.xml`) may also be overridden by the `weblogic.Deployer -name` option or by the manifest application name attribute.

The naming precedence is as follows, starting with the highest precedence:

- The `-name weblogic.Deployer` option.
- The name specified in the manifest file.
- The name specified in the deployment descriptor.
- A computed name based on the deployment source file.

If a provided or computed application or module name is already in use, now WebLogic Server will modify the name to make it unique by adding a numerical suffix.

See the following section, Understanding Application Naming Requirements for information on application naming requirements; See Deploying Applications and Modules with weblogic.Deployer to specify a non-default deployment name.

Understanding Application Naming Requirements

In order to successfully deploy an application to WebLogic Server, the application name must be valid.

Application naming requirements are as follows:

- Application names must only contain the following characters:
  - a-z
  - A-Z
  - 0-9
  - _ (underscore)
  - - (hyphen)
  - . (period)

  No additional characters are allowed in application names.
- Application names that contain the "." character must contain at least one additional different character; "." and ".." are not valid application names.
- Application names must be less than 215 characters in length.

Understanding Deployment Version Strings

In addition to a deployment name, an application or module can also have an associated version string. The version string distinguishes the initial deployment of the application from subsequent redeployed versions.

For example, you may want to later update the application to fix problems or add new features. In production systems, it is critical to maintain a version string for both the initial and subsequent deployments of an application. Doing so allows you to update and redeploy an application version without interrupting service to existing clients. See Redeploying Applications in a Production Environment for more information.

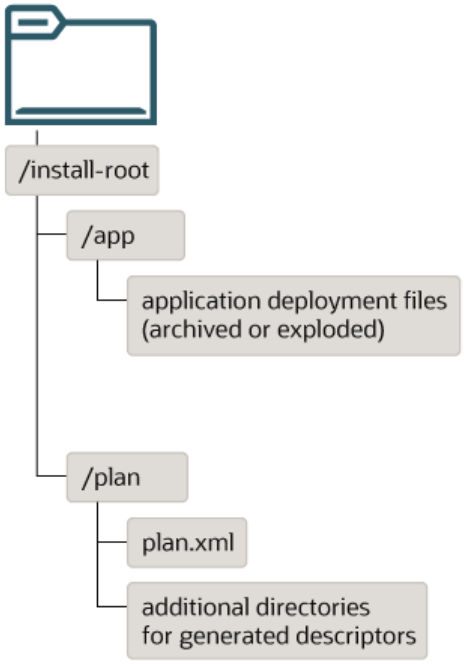
The version string is specified in the manifest file for the application, and should be provided by your development team along with the other deployment files. Assigning Application Versions in *Developing Applications for Oracle WebLogic Server* describes the conventions for specifying the version string.

Creating an Application Installation Directory

The application installation directory separates generated configuration files from the core application files, so that configuration files can be easily changed or replaced without disturbing the application itself. The directory structure also helps you to organize and maintain multiple versions of the same application deployment files.

The following figure shows the application installation directory hierarchy for storing a single version of a deployable application or module.

Figure 2-1 Application Installation Directory



Oracle recommends copying all new production deployments into an application installation directory before deploying to a WebLogic Server domain. Deploying from this directory structure helps you easily identify all of the files associated with a deployment unit—you simply deploy the installation root using the WebLogic Remote Console, and the Console automatically locates associated files such as deployment plans and WebLogic Server deployment descriptors that were generated during configuration.

- Steps for Creating an Application Installation Directory

Steps for Creating an Application Installation Directory

To create an application installation directory:

1. Choose a top-level directory where you want to store deployment files for applications and modules on your system. Follow these best practices:
   - Do not store deployment files within a WebLogic Server domain directory.
   - Use source control if available to maintain deployment source files.
   - If possible, store deployment files in a directory that is accessible by the Administration Server and Managed Servers in your domain.

   The instructions that follow use the sample deployment directory, `c:\deployments\production`.
2. Create a dedicated subdirectory for the application or module you want to deploy:

   ```
   mkdir c:\deployments\production\myApplication
   ```
3. Create a subdirectory beneath the application directory to designate the version of the application you are deploying. Name the subdirectory using the exact version string of the application. For example:

   ```
   mkdir c:\deployments\production\myApplication\Beta
   ```

4. The version subdirectory will become the installation root directory from which you deploy the directory. Create subdirectories named `app` and `plan` under the version subdirectory:

```
mkdir c:\deployments\production\myApplication\Beta\app
mkdir c:\deployments\production\myApplication\Beta\plan
```

> **Note:**
>
> If you have more than one deployment plan associated with the application, create one `\plan` subdirectory for each plan. For example, if you have two deployment plans associated with the Beta version of the application myApplication, you would create two `\plan` subdirectories. For instance:
>
> - `mkdir c:\deployments\production\myApplication\Beta\plan1`
> - `mkdir c:\deployments\production\myApplication\Beta\plan2`

5. Copy your application source deployment files into the `\app` subdirectory. If you are deploying from an archive file, simply copy the archive file, as in:

```
cp c:\downloads\myApplication.ear c:\deployments\production\myApplication\Beta\app
```

   If you are deploying from an exploded archive directory, copy the complete exploded archive directory into `\app`:

```
cp -r c:\downloads\myApplication c:\deployments\production\myApplication\Beta\app
```

   This results in the new directory, `c:\deployments\production\myApplication\Beta\app\myApplication`.

6. If you have one or more deployment plans for the application, copy them into the `\plan` subdirectories.

   If you have one deployment plan for the application:

```
cp c:\downloads\myApplicationPlans\Plan.xml
c:\deployments\production\myApplication\Beta\plan
```

   If you have two deployment plans for the application:

```
cp c:\downloads\myApplicationPlans\plan1.xml
c:\deployments\production\myApplication\Beta\plan1
cp c:\downloads\myApplicationPlans\plan2.xml
c:\deployments\production\myApplication\Beta\plan2
```

> **Note:**
>
> If you do not have an existing deployment plan, you can create one using the WebLogic Remote Console as described in Configuring Applications for Production Deployment. The WebLogic Remote Console stores newly-generated deployment plans in the `\plan` subdirectory of the application installation directory.

7. To install the application using the WebLogic Remote Console, select the application installation directory. By default, the WebLogic Remote Console will use a plan named `Plan.xml`, if one is available in the `\plan` subdirectory. The WebLogic Remote Console does not identify plans in subdirectories other than the `\plan` subdirectory; in other words,

plans in \plan1 or \plan2 subdirectories are not identified by the WebLogic Remote Console. Therefore, if multiple plans for your application are available, you must indicate, in config.xml, the plan you would like to use. See Configuring Applications for Production Deployment. For information on config.xml, see *Creating WebLogic Domains Using the Configuration Wizard*.

After installing the application, you can configure, deploy, or distribute the application as necessary.

> **Note:**
>
> You cannot specify an application installation directory when using the weblogic.Deployer tool, and the tool does not use an available plan.xml file by default. You must specify the actual deployment file or files and plan to use for deployment. See Deploying Applications and Modules with weblogic.Deployer.

Using FastSwap Deployment to Minimize Redeployment

Today's Web application developers expect to make changes to a deployed application and see those changes immediately by refreshing the browser.

On the Jakarta EE side, developers typically have to go through an Edit -> Build -> Deploy -> Test cycle to see their changes in action. These steps, along with the many required descriptor elements, makes developing applications with Jakarta EE seem complex and top-heavy.

Among these steps, the build and deploy cycles are necessitated by Java and by the application server being employed. IDEs are trying to make the edit and build steps seamless by providing incremental compilation support. On the server side, the WebLogic Server FastSwap deployment feature makes the deploy and test cycles just as seamless.

- How FastSwap Deployment Works
- Supported FastSwap Application Configurations
- Enabling FastSwap In Your Application
- Overview of the FastSwap Process
- Application Types and Changes Supported with FastSwap
- Limitations When Using FastSwap
- Handling Unsupported FastSwap Changes

How FastSwap Deployment Works

Jakarta EE introduced the ability to redefine a class at runtime without dropping its classloader or abandoning existing instances. This allowed containers to reload altered classes without disturbing running applications, vastly speeding up iterative development cycles and improving the overall development and testing experiences. The usefulness of the Java EE dynamic class redefinition is severely curtailed, however, by the restriction that the shape of the class – its declared fields and methods – cannot change. The purpose of FastSwap is to remove this restriction in WebLogic Server, allowing the dynamic redefinition of classes with new shapes to facilitate iterative development.

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, auto compile, and then see the effects immediately.

Supported FastSwap Application Configurations

The following application configurations are supported when using FastSwap deployment:

- FastSwap is only supported when WebLogic Server is running in development mode. It is 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. Examples are as follows:
  - When a web application is deployed as an archived WAR within an EAR, modifications to any of the classes are not picked up by the FastSwap agent.
  - Within an exploded web application, modifications to Java classes are only supported in the `WEB-INF/classes` directory; the FastSwap agent does not pick up changes to archived JARs residing in `WEB-INF/lib`.

Enabling FastSwap In Your Application

To enable FastSwap in your application, add the following element to the `weblogic-application.xml` file.

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

For more information on the `weblogic-application.xml` elements, see Enterprise Application Deployment Descriptor Elements in *Developing Applications for Oracle WebLogic Server*.

FastSwap can also be enabled for a standalone Web application by adding the `<fast-swap>` element to the `weblogic.xml` file. For more information on the `weblogic.xml` elements, see weblogic.xml Deployment Descriptor Elements in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

Overview of the FastSwap Process

The following steps describe how the FastSwap deployment process works:

1. After FastSwap is enabled at the descriptor level, an appropriate classloader is instantiated when the application is deployed to WebLogic Server.
2. Open a browser to see the application at work. Modify (add, edit, or delete) the methods and classes as desired (see Limitations When Using FastSwap), and then compile them.

   Oracle recommends that you use an IDE such as Eclipse or IntelliJ and set the compile-on-save option so that the Java files are compiled on saving. Also note that the FastSwap agent does not compile Java files.
3. Refresh the browser or send a new request to the application.

   The FastSwap agent tries to find all the classes that have been modified since the last iteration by looking at all directories in the classpath. Considering an exploded application

with a single web application, the following directories are examined for any class file modifications based on their timestamps:

```
ExampleApp/APP-INF/classes
ExampleApp/webapp/WEB-INF/classes
```

The FastSwap agent redefines the modified classes in the application and then serves the request.

- Using Ant with the JMX Interface

Using Ant with the JMX Interface

For *headless* applications (that is, applications not fronted by a web application), class redefinition can be explicitly initiated using the JMX interface. An Ant task that uses the JMX interface can be used to initiate class redefinition, as shown in following Ant FastSwapTask example.

Example 2-1 Using Ant with the JMX Interface

```
<project name='MyProject' default='all' >
  <taskdef name='fast-swap'  classname='com.bea.wls.redef.ant.FastSwapTask'/>
  <target name='all'>
    <!--
      Redefine classes which have changed since they were last loaded.
      Required parameters:
       adminUrl: Connection url
           user: User name
       password: User password
         server: Managed server name
      application: Deployed application name
      Optional parameters:
          module: Name of the module within the application.
                  If not specified, all modules within the application
                  will be processed.
       failonerror: Default=true. If true, task will fail if fast-swap failed.
                  Otherwise, a message will be displayed and the task will
                    return success.
         timeout: Default=300. Timeout in seconds.
      classnames: Comma separated list of classnames to process. If not
                  specified, all classes within specified modules (if any)
                    in the application will be considered.
    -->
    <fast-swap
      adminUrl='t3://localhost:7001'
      user='weblogic'
      password='weblogic'
      server='myserver'
       application='SimpleApp'
      module='SimpleAppCookie'
      failonerror='false'
      timeout='30'
      classnames='examples.servlets.CookieCounter1,
                  examples.servlets.CookieCounter2,
                    examples.servlets.CookieCounter'
    />
  </target>
</project>
```

Application Types and Changes Supported with FastSwap

FastSwap is supported with POJOs (JARs), Web applications (WARs) and enterprise applications (EARs) deployed in an exploded format. FastSwap is not supported with resource adapters (RARs).

The following types of changes are supported with FastSwap:

- Addition of static methods
- Removal of static methods
- Addition of instance methods
- Removal of instance methods
- Changes to static method bodies
- Changes to instance method bodies
- Addition of static fields
- Removal of static fields
- Addition of instance fields
- Removal of instance fields

The following table lists detailed change types supported with FastSwap:

Table 2-1 Supported Application Types and Changes

Scope	Java Change Type	Supported	Notes
Java Class	Add method	Yes	Addition of the `finalize` method is not supported.
Instance (non-abstract)	Remove method	Yes	Addition of the `finalize` method is not supported.
	a) Add field	Yes	Not Applicable (N/A)
	b) Remove field	Yes	
	c) Change method body	Yes	
	d) Add constructor	Yes	
	e) Remove constructor	Yes	
	f) Change field modifiers	Yes	
	g) Change method modifiers	Yes	
Class-level (static)	Add method	Yes	N/A
	Remove method	Yes	
	Change body method	Yes	
Class Hierarchy Changes	Change list of implemented interfaces	No	N/A
	Change extends "SuperClass"	No	
Abstract Java Class	Add abstract method	Yes	N/A
	Delete abstract method	Yes	

Table 2-1 (Cont.) Supported Application Types and Changes

Scope	Java Change Type	Supported	Notes
	All other supported changes (a–g) listed in `Instance`	Yes	
"final" Java Class	Same supported changes (a–g) listed in `Instance`	Yes	N/A
"final" Java Method	Same supported changes (a–g) listed in `Instance`	Yes	N/A
"final" Java Field	Same supported changes (a–g) listed in `Instance`	Yes	N/A
Enum	Add constants	No	N/A
	Remove constants	No	
	Add/remove methods	No	
Anonymous Inner Class	Add/remove fields	NA	Not supported by the Java language
	Add/remove methods	No	N/A
Static Inner Class	Same supported changes (a–g) listed in `Instance`	Yes	N/A
Member Inner Classes (non-static inner classes)	Same supported changes (a–g) listed in `Instance`	Yes	N/A
Local Inner Classes	Same supported changes (a–g) listed in `Instance`	Yes	N/A
Java Interface	Add method	Yes	N/A
Java Reflection	Access existing fields/methods	Yes	N/A
	Access new methods	No	New methods are not seen using Reflection and some synthetic methods are exposed.
	Access new fields	No	New fields are not seen using Reflection.
Annotations on Classes	Add or remove method/field annotations	No	N/A
Annotation Type	Add or remove methods/attributes	No	N/A
Exception Classes	Same supported changes (a–g) listed in `Instance`	Yes	N/A
EJB Interface	Add/remove methods	No	Changes to EJB interfaces involve Reflection, which is not fully supported.
EJB 3.0 Session/MDB EJB Implementation Class	Add/remove methods	No	Any support classes referenced by the EJB classes can be modified.
	Add/remove fields	No	N/A
EJB 3.0 EntityBean	Add/remove methods	No	Any support classes referenced by the EJB classes can be modified.
	Add/remove fields	No	N/A
EJB Interceptors	Add/remove methods	No	Any support classes referenced by the EJB classes can be modified.
	Add/remove fields	No	N/A

Limitations When Using FastSwap

The following limitations apply when using FastSwap deployment:

- Java reflection results do not include newly added fields and methods, and include the removed fields and methods. As a result, use of the reflection API on the modified classes can result in undesired behavior.
- Changing the hierarchy of an already existing class is not supported by FastSwap. For example, either changing the list of implemented interfaces of a class or changing the superclass of a class, is not supported.
- Addition or removal of Jakarta annotations is not supported by FastSwap, because this is tied to the reflection changes mentioned above.
- Addition or removal of methods on EJB interfaces is not supported by FastSwap because an EJB compilation step is required to reflect the changes at runtime.
- Addition or removal of constants from enums is not supported.
- Addition or removal of the finalize method is not supported.
- When you change a field name, the object state is not retained. This type of change occurs as follows: the field with the old name is deleted and a field with the new name is added. As such, any state in the old field is not carried over to the renamed field. You should expect an instance value to be reset when you change a field name.

Handling Unsupported FastSwap Changes

When FastSwap is enabled, after you recompile a class, FastSwap attempts to redefine classes in existing classloaders. If redefinition fails because your changes fall outside the scope of supported FastSwap changes, the JVM throws an `UnsupportedOperationException` in the WebLogic Server window and in the server log file. Your application will not reflect the changes, but will continue to run.

To implement your changes, you can redeploy the application or affected modules (partial redeploy), depending on the application type and the extent of your changes.

Best Practices for Preparing Deployment Files

Follow Oracle-recommended best practices when preparing applications and modules for deployment.

- Regardless of whether you deploy an archive file or exploded archive directory, store the deployment files in an installation directory for the application, as described in Creating an Application Installation Directory. Using an installation directory simplifies the deployment process, because the WebLogic Remote Console understands where to locate deployment and configuration files.
- Manage the entire application installation directory in a source control system, so you can easily revert to previous application versions if necessary.

3

Configuring Applications for Production Deployment

Learn how to configure your applications for deployment to a production WebLogic Server environment.
This chapter includes the following sections:

- Understanding the Deployment Configuration Process
  When an administrator or deployer receives a new application, or a new version of an application, from development or quality assurance teams, the application is usually configured for a development or testing environment.
- Typical Deployment Configuration Workflows
  Deployment plans enable you to define a convenient, repeatable workflow for configuring an application for deployment to multiple WebLogic Server environments. A configuration workflow for production applications requires cooperation between your development and design teams, which create and package the deployable application, and the administrator or deployer for each target WebLogic Server environment.
- Creating a New Deployment Plan to Configure an Application
  The WebLogic Remote Console automatically generates (or updates) a valid XML deployment plan for an application when you interactively change deployment properties for an application that you have installed to the domain. You can use the generated deployment plan to configure the application in subsequent deployments, or you can generate new versions of the deployment plan by repeatedly editing and saving deployment properties.
- Understanding Deployment Plan Contents
- Using an Existing Deployment Plan to Configure an Application
  Applications that you receive for deployment may come with varying levels of configuration information.
- Generic File Loading Overrides
  Generic file loading overrides let you place application-specific files to be overridden into an optional subdirectory (named `/AppFileOverrides`) in the existing plan directory structure. The presence or absence of this subdirectory controls whether file overrides are enabled for the deployment.
- Additional Configuration Tasks
  The referenced information describes additional deployment configuration tasks.
- Best Practices for Managing Application Configuration

Understanding the Deployment Configuration Process

When an administrator or deployer receives a new application, or a new version of an application, from development or quality assurance teams, the application is usually configured for a development or testing environment.

The application may use specific resource names and performance tuning settings that match the available resources on the target servers used in the development or QA environments where the application was last deployed. Because development and testing environments can

be significantly different from the production environment in which the application is ultimately deployed, an administrator must configure the application to use resource names and performance tuning parameters that are valid and appropriate for the production environment.

- Deployment Configuration Life Cycle
- Understanding Application Deployment Descriptors
- Understanding WebLogic Server Deployment Plans
- Goals for Production Deployment Configuration

Deployment Configuration Life Cycle

Deployment configuration for an application can occur at several points in the life cycle of an application. Each phase of deployment configuration typically involves creating and working with different deployment files:

1. Development configuration—During development, a programmer creates Jakarta EE deployment descriptors for an application or module. The programmer also creates WebLogic Server deployment descriptors to configure the application for deployment to a WebLogic Server development environment. See *Developing Applications for Oracle WebLogic Server*.

   > **Note:**
   >
   > Applications developed outside of the WebLogic Server development environment (for example, a sample or third-party Jakarta EE application such as PetStore) may include only Jakarta EE descriptors.

2. Export configuration—Before releasing an application from development, a programmer or designer may optionally export the application's deployment configuration to a WebLogic Server deployment plan. Exporting a configuration creates deployment plan variables for all or a subset of the deployment properties already defined by a developer in the application's WebLogic Server deployment descriptor files. See Exporting an Application for Deployment to New Environments.

   Exporting an application helps deployers in other areas of the organization (such as engineers on the QA team or production administrators) easily deploy the application to environments that differ from the programmer's development environment. The ideal deployment plan includes all of the properties that a deployer needs to change before deploying the application in a new environment.

3. Deployment-time configuration—An administrator or deployer configures the application before deploying the application into the target environment. Deployment-time configuration may use:

   - The same WebLogic Server deployment configuration and deployment plan created during development.
   - Modified versions of the development configuration and deployment plan.
   - A custom deployment plan that the deployer previously created for the environment, depending on the deployment configuration workflow for your organization.

   See Deploying Applications and Modules with weblogic.Deployer.

4. Post-deployment configuration—After an application has been deployed to a target environment, an administrator or deployer can reconfigure the application by redeploying

with a new deployment plan or by using the WebLogic Remote Console to update and redeploy an existing deployment plan. See Redeploying Applications in a Production Environment and Managing Deployed Applications.

Because deployment configuration is performed by different people at different points in the life cycle of an application, administrators, deployers, and developers need to work together to create a repeatable configuration workflow for their organization. See Typical Deployment Configuration Workflows.

Understanding Application Deployment Descriptors

The basic deployment configuration for an application is defined in multiple XML documents, known as deployment descriptors, that are included as part of the application archive file that you receive for deployment. Deployment descriptor files fall into two separate categories:

- *Jakarta EE deployment descriptors* define the fundamental organization and behavior of a Jakarta EE application or module, regardless of where the application is deployed. Each Jakarta EE application and module requires a specific Jakarta EE deployment descriptor as defined in the Jakarta EE specification.
- *WebLogic Server deployment descriptors* define the resource dependencies and tuning parameters that an application uses in a specific WebLogic Server environment.

For the purposes of a production deployment, you should treat both the Jakarta EE and WebLogic Server deployment descriptors as part of the application's source code, which is owned by your development team. Do not edit application deployment descriptors in order to configure an application for deployment to a production environment. Instead, persist configuration changes into a WebLogic Server *deployment plan*, which is described in the next section.

> **Note:**
>
> You cannot use deployment plans with applications using DTD-based deployment descriptors. You must upgrade the application to use schema-based descriptors.

Understanding WebLogic Server Deployment Plans

As previously discussed, a WebLogic Server deployment plan is an optional XML document that you use to configure an application for deployment to a specific WebLogic Server environment. A deployment plan specifies setting deployment property values that would normally be defined in an application's WebLogic Server deployment descriptors, or overrides property values that are already defined in a WebLogic Server deployment descriptor. When exporting an application, the deployment plan typically acts to override selected properties in the WebLogic Server deployment descriptors you created during development.

> **Note:**
>
> You can configure or override any Jakarta EE or WebLogic-specific deployment descriptors, except items annotated as transient.
>
> You can update deployment plans using JSR 88 DConfigBeans setter methods to change the values of the descriptor elements and the remove methods to remove descriptor elements. Also, you can call the setter and remove methods from the descriptor bean tree returned by `getDescriptorBean()` for both standard and WebLogic-specific descriptors. These methods result in overrides in the plan. See Configuring Applications for Deployment in *Deploying Applications with the WebLogic Deployment API*.

Typically, deployment plans are created by developers along with the associated application files, then distributed to the administrator or another deployer, who updates the plan for a particular environment (such as staging, testing, or production). The deployment plan is stored *outside* of an application archive or exploded archive directory. As a best practice, Oracle recommends storing each deployment plan for a single application in its own `plan` subdirectory of the application's root directory (See Creating an Application Installation Directory).

Deployment plans help the administrator easily modify an application's WebLogic Server configuration for deployment to multiple, differing WebLogic Server environments *without* modifying the deployment descriptor files included in the application archive. Configuration changes are applied by adding or changing variables in the deployment plan, which define both the location of the WebLogic Server descriptor properties to change and the value to assign to those properties. Administrators deploying an application need only change the deployment plan—the original deployment files and deployment descriptors remain unchanged as shown in Figure 3-1. To determine the deployment configuration workflow for your environment, see Typical Deployment Configuration Workflows.

Figure 3-1 WebLogic Server Deployment Plan

Goals for Production Deployment Configuration

For the administrator, the primary goal of configuring an application for production deployment is to generate a new deployment plan that is valid and appropriate for the target WebLogic Server environment. Specifically, the deployment plan must resolve all external resources references for the application to refer to valid resources available in the target environment. If the application's configuration does not define to valid external resources for the target servers, the application cannot be deployed.

A deployment plan can optionally define or override WebLogic Server tuning parameters, to make ideal use of resources in the target environment. Defining tuning parameters is not required in order to successfully deploy an application. If an application's deployment descriptors and deployment plan do not define tuning parameters, WebLogic Server uses default values.

Typical Deployment Configuration Workflows

Deployment plans enable you to define a convenient, repeatable workflow for configuring an application for deployment to multiple WebLogic Server environments. A configuration workflow for production applications requires cooperation between your development and design teams, which create and package the deployable application, and the administrator or deployer for each target WebLogic Server environment.

The ideal deployment configuration workflow for your organization is determined by:

- Number of different environments in which you deploy the same application
- Difference in resources provided by each target environment
- Frequency with which each target environment changes
- Frequency with which the application's Jakarta EE configuration changes
- Ownership requirements for configuration information in different areas of your organization

The sections that follow describe common deployment configuration workflows for managing deployment plans and deploying applications to multiple WebLogic Server domains.

- Application with Single Deployment Plan
- Application with Multiple Deployment Plans

Application with Single Deployment Plan

Organizations that know the exact configuration of different deployment environments can use a single, well-defined deployment plan to deploy an application to multiple WebLogic Server domains. The single deployment plan configuration workflow works in the following way:

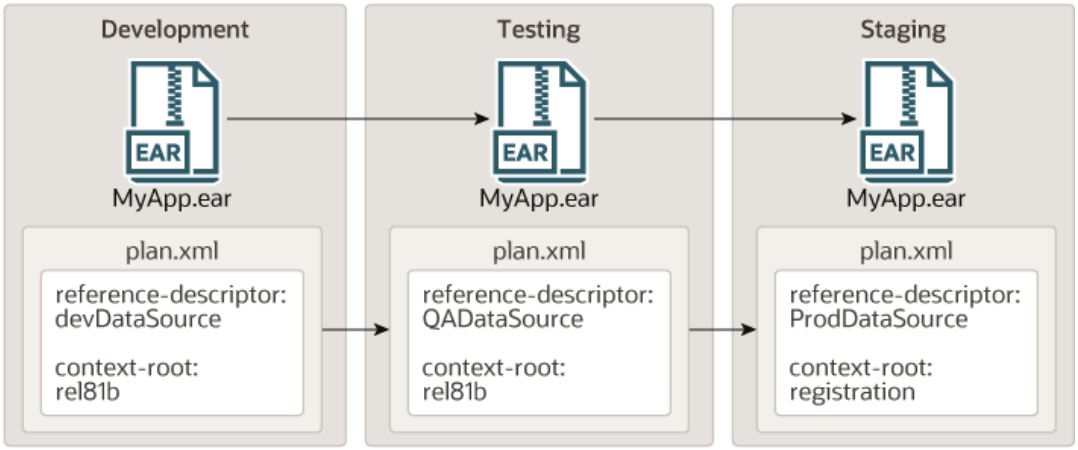
1. The development team, cooperating with administrators and deployers, creates a primary deployment plan for use with all target environments. The number of target environments will vary depending on your organizational structure. Common deployment environments include one or more Quality Assurance (QA) or testing domains, staging domains, and production domains.

   The deployment plan that the team creates at this phase defines variables for all configuration properties that are known to differ between each target environment. For example, the plan might define empty variables for resource names that differ between environments and must be configured before the application can be deployed. The plan may also define default values for common tuning parameters that deployers may want to change in their environments.

   For more information about creating a deployment plan during development, see Exporting an Application for Deployment to New Environments.

2. When a version of the application is ready to be released, the development team packages the application deployment files and delivers both the deployment files and a primary deployment plan to deployers for each target environment.
3. Each deployer uses the WebLogic Remote Console to install the application and identify the deployment plan to use for configuration. The WebLogic Remote Console validates the overall deployment configuration based on the resources available in the target domain. The Console then presents a list of configurable properties defined in the plan (as well as any invalid properties) to the deployer for editing.

Figure 3-2 Single Deployment Plan Workflow



4. The deployer uses the WebLogic Remote Console to interactively configure properties that were defined in the deployment plan. Deployment plan variables that have null values, or invalid values for the target WebLogic Server instances or clusters, must be configured before the application can be deployed. Deployment plan variables that already have valid values need not be changed before deployment.

   Deployers in each environment agree to limit their configuration changes to those properties defined in the deployment plan. If additional configuration changes are required, the deployer must communicate those requirements to the development or design team that modifies the primary deployment plan.

- Benefits of a Single Deployment Plan Workflow
- Single Deployment Plan Ownership and Limitations

Benefits of a Single Deployment Plan Workflow

Using the single deployment plan workflow provides the following benefits:

- It enables the development or design team to control the deployment configuration of the application.
- It reduces the number of configuration decisions that a deployer must make when deploying the application to a target environment.
- It minimizes the number of configuration files associated with the application, making it easy to store deployment configuration information in a source control system.

In general, you would use a single deployment plan workflow if your organization has a few, well-understood target environments, and you want to easily replicate a standardized deployment configuration in each environment.

Single Deployment Plan Ownership and Limitations

The single deployment plan workflow assumes that the development or design team maintains ownership of the deployment plan, and that deployers limit their plan changes to those variables defined in the plan. If the deployer modifies only those properties defined in the deployment plan, their changes are written back to the same deployment plan as updates to the variables.

However, WebLogic Server imposes no restrictions on the configuration properties that a deployer can modify using the WebLogic Remote Console. If a deployer configures

deployment properties that were not originally defined in a plan, the Console generates a new deployment plan having the additional variable entries, and uses the new plan for deployment or redeployment operations. This can lead to a situation where the deployer uses a deployment plan that is drastically different from the primary deployment plan owned by the development team.

To incorporate new changes into the primary deployment plan, the deployer retrieves the new, customized deployment plan created by the Console. Ideally, those changes should be applied to the primary deployment plan.

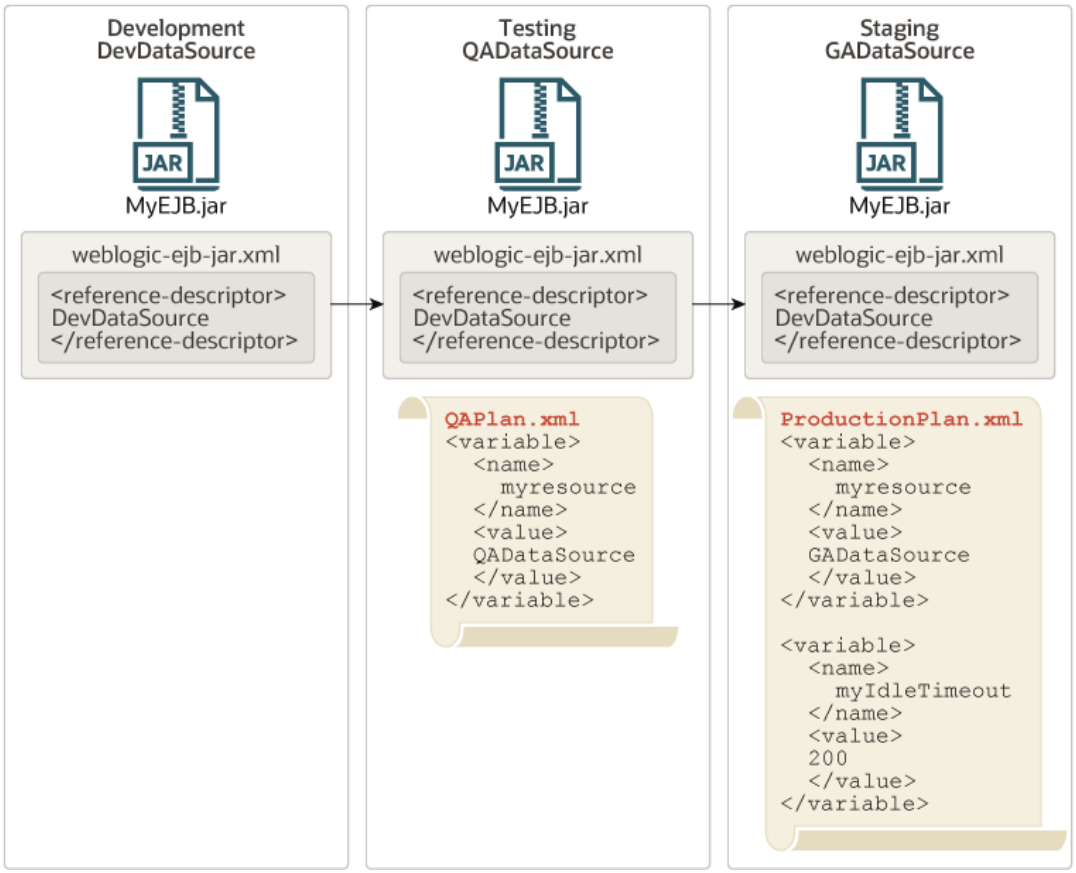
Application with Multiple Deployment Plans

Organizations that have numerous deployment environments that frequently change should use a configuration workflow with multiple deployment plans. In a multiple deployment plan workflow, each deployment plan is owned by the deployer of the application rather than the development team. The multiple deployment plan configuration workflow works in the following way:

1. The development team releases a version of the packaged application deployment files (containing Jakarta EE and WebLogic Server descriptors). The development team may or may not include a template deployment plan with exported variables for resource definitions or common tunable parameters.
2. Before deploying the application, each deployer generates a custom deployment plan to configure the application for their target environment.

   A custom deployment plan can be created by starting with a template deployment plan (or no deployment plan) and making changes to the application's deployment configuration using the WebLogic Remote Console. See Modify a Deployment Plan in the *Oracle WebLogic Remote Console Online Help*.
3. After defining the deployment configuration for their environment, each deployer retrieves their custom deployment plan and maintains it for future deployments of the application. Oracle recommends storing custom configuration plans in a source control system so that new versions can be tracked and reverted to if necessary.

Figure 3-3 Multiple Deployment Plan Workflow



4. For subsequent releases of the application, each deployer uses their customized deployment plan to configure the application for deployment. Using the customized plan allows deployers to perform deployments with the `weblogic.Deployer` or automate deployments using WLST.

- Benefits of a Multiple Deployment Plan Workflow
- Multiple Deployment Plan Ownership and Limitations

Benefits of a Multiple Deployment Plan Workflow

Using the multiple deployment plan workflow provides the following benefits:

- It enables the administrator or deployer to manage both the application configuration and environment configuration in tandem.
- It enables deployers to automate the deployment process by using a custom plan that fully configures the application for their application.

In general, you would use a multiple deployment plan workflow if your organization has many deployment environments that change frequently, making it difficult or impossible to maintain a single primary deployment plan.

Multiple Deployment Plan Ownership and Limitations

The multiple deployment plan workflow assumes that the deployer or administrator (rather than the programming or design team) owns and maintains the deployment configuration for an application. It also assumes that the basic Jakarta EE configuration of the application rarely

changes, because certain Jakarta EE configuration changes would render a deployer's custom configuration plans invalid. For example, if a module in an enterprise application is added, removed, or changed, custom deployment plans referencing the module would become invalid. In this case, each deployer would need to re-create a custom plan by configuring the application using the WebLogic Remote Console.

Creating a New Deployment Plan to Configure an Application

The WebLogic Remote Console automatically generates (or updates) a valid XML deployment plan for an application when you interactively change deployment properties for an application that you have installed to the domain. You can use the generated deployment plan to configure the application in subsequent deployments, or you can generate new versions of the deployment plan by repeatedly editing and saving deployment properties.

> **Note:**
>
> `weblogic.PlanGenerator` also enables you to generate a basic WebLogic Server deployment plan for applications that have only Jakarta EE deployment descriptors. See weblogic.PlanGenerator Command-Line Reference.

Generating a deployment plan using the WebLogic Remote Console involves the steps in the following topics.

- Preparing the Deployment Files
- Installing the Application Archive
- Saving Configuration Changes to a Deployment Plan

Preparing the Deployment Files

Use the following directions to prepare your application for deployment.

1. Create a new root directory and `app` and `plan` subdirectories for your application. For example:

   ```
   mkdir c:\sample_root
   mkdir c:\sample_root\app
   mkdir c:\sample_root\plan
   ```

2. Save your application to the `c:\sample_root\app` directory you created in Step 1.

Installing the Application Archive

Use the WebLogic Remote Console to install a new application for configuration and deployment to a new WebLogic Server environment. After you install the application or module and select deployment targets, the deployment files are available in the WebLogic Server domain and can be configured, distributed, and deployed as necessary.

Follow the steps in Install an Application in the *Oracle WebLogic Remote Console Online Help* to install your application to your WebLogic Server domain.

Saving Configuration Changes to a Deployment Plan

Use the WebLogic Remote Console to edit any deployment configuration properties for the application you installed in Installing the Application Archive and save the configuration to a deployment plan. See Specify Deployment Properties in the *Oracle WebLogic Remote Console Online Help*.

1. On the Configuration page, edit one or more configuration properties as needed for your application. For example, change the Session Invalidation Interval to 80 seconds and the Session Timeout to 8000 seconds.
2. Click **Save** to save your changes. The WebLogic Remote Console stores your configuration changes to a new deployment plan. If you deployed your application from a root directory, the WebLogic Remote Console automatically places the new deployment plan in the `\plan` subdirectory of the root directory. For example, `c:\sample_root\plan\Plan.xml`.

Understanding Deployment Plan Contents

Examine a sample deployment plan. Example 3-1 shows a sample deployment plan generated by the steps in Creating a New Deployment Plan to Configure an Application.

Example 3-1 Sample Deployment Plan

```
<deployment-plan xmlns="http://xmlns.oracle.com/weblogic/deployment-plan">
  <application-name>sample_root</application-name>
    <variable-definition>
    <variable>
      <name>SessionDescriptor_InvalidationIntervalSecs_11029744771850</name>
      <value>80</value>
    </variable>
    <variable>
      <name>SessionDescriptor_TimeoutSecs_11029744772011</name>
      <value>8000</value>
    </variable>
    </variable-definition>
  <module-override>
    <module-name>myApp.ear</module-name>
    <module-type>ear</module-type>
    <module-descriptor external="false">
      <root-element>weblogic-application</root-element>
      <uri>META-INF/weblogic-application.xml</uri>
    </module-descriptor>
    <module-descriptor external="false">
      <root-element>application</root-element>
      <uri>META-INF/application.xml</uri>
    </module-descriptor>
  </module-override>
  <module-override>
    <module-name>myWebApp.war</module-name>
    <module-type>war</module-type>
    <module-descriptor external="false">
      <root-element>weblogic-web-app</root-element>
      <uri>WEB-INF/weblogic.xml</uri>
      <variable-assignment>
        <name>SessionDescriptor_InvalidationIntervalSecs_11029744771850</name>
        <xpath>/weblogic-web-app/session-descriptor/invalidation-interval-secs</xpath>
      </variable-assignment>
      <variable-assignment>
```

```
        <name>SessionDescriptor_TimeoutSecs_11029744772011</name>
        <xpath>/weblogic-web-app/session-descriptor/timeout-secs</xpath>
      </variable-assignment>
    </module-descriptor>
    <module-descriptor external="false">
      <root-element>web-app</root-element>
      <uri>WEB-INF/web.xml</uri>
    </module-descriptor>
  </module-override>
  <module-override>
    <module-name>sample_root</module-name>
    <module-type>ear</module-type>
    <module-descriptor external="false">
      <root-element>weblogic-application</root-element>
      <uri>META-INF/weblogic-application.xml</uri>
    </module-descriptor>
    <module-descriptor external="false">
      <root-element>application</root-element>
      <uri>META-INF/application.xml</uri>
    </module-descriptor>
  </module-override>
  <config-root>C:\sample_root\plan</config-root>
</deployment-plan>
```

The basic elements in the deployment plan serve the following functions:

- `deployment-plan` encapsulates all of the deployment plan's contents.
- `application-name` corresponds to the deployment name for the application or module.
- `variable-definition` defines one or more `variable` elements. Each `variable` defines the `name` of a variable used in a plan and a `value` to assign (which can be null). The sample plan shown in Example 3-1 contains variable definitions for the changes you made to the Session Invalidation Interval and Session Timeout properties.
- `module-override` elements define each module name, type, and deployment descriptor that the deployment plan overrides. A `module-descriptor` element can optionally contain a `variable-assignment` which identifies a variable name used to override a property in the descriptor, and the exact location within the descriptor where the property is overridden.

  The sample plan shown in Example 3-1 contains module override elements for the enterprise application, the embedded Web application, and the enclosing root directory. The module-descriptor entry for the `weblogic.xml` descriptor file contains two `variable-assignment` elements that override the property values for the Session Invalidation Interval and Session Timeout properties, as changed in the example in Saving Configuration Changes to a Deployment Plan.

  By default, the values in `variable-assignment` elements are added to the values that are already defined in the descriptor. You can change this behavior and cause the `variable-assignment` element to replace or remove the values that are defined in the descriptor by setting the `operation` sub-element in the `variable-assignment` element to the value `replace` or `remove`, respectively.

  For example, suppose that in the `ejb-jar.xml` file, a developer created a policy to allow access only to the security role named `ejbRole`.

  ```
  ...
  <assembly-descriptor>
  <security-role>
  <role-name>ejbRole</role-name>
  </security-role>
  <method-permission>
  ```

```
<role-name>ejbRole</role-name>
<method>
<ejb-name>ejb.SearchHandlerWrapperEJB</ejb-name>
<method-name>*</method-name>
</method>
</method-permission>
</assembly-descriptor>
...
```

And, the `security-role-assignment` element in weblogic-ejb-jar.xml, `ejbRole` is mapped to the principal named `user1`.

```
...
<security-role-assignment>
<role-name>ejbRole</role-name>
<principal-name>user1</principal-name>
</security-role-assignment>
...
```

If you want to use a deployment plan to override the `security-role-assignment` element defined in `weblogic-ejb-jar.xml`, so that `ejbRole` is mapped to `user2` instead of `user1`, you could achieve the desired override behavior by setting appropriate values for the `variable`, `variable-assignment`, and `operation` elements in the deployment plan. Make sure to set the value of `operation` to `replace`:

```
...
<variable> <name>SecurityRoleAssignment_ejbRole_PrincipalNames_11168815313911</name>
<value>user2</value>
</variable>

<variable-assignment>
<name>SecurityRoleAssignment_ejbRole_PrincipalNames_11168815313911</name>
<xpath>/weblogic-ejb-jar/security-role-assignment/[role-name="ejbRole"]/principal-
name</xpath>
<operation>replace</operation>
</variable-assignment>
```

For more information about the contents of a WebLogic Server deployment plan, see `http://xmlns.oracle.com/weblogic/deployment-plan/1.01/deployment-plan.xsd`.

Using an Existing Deployment Plan to Configure an Application

Applications that you receive for deployment may come with varying levels of configuration information.

If you have an existing deployment plan for an application, simply prepare the application as described in Preparing the Deployment Files and place the deployment plan in the `plan` subdirectory of the application root. Then install the application using the instructions in Installing the Application Archive. The WebLogic Remote Console automatically uses a deployment plan named `Plan.xml` in the `\plan` subdirectory of an application root directory if one is available. If multiple plans are available for your application, they are placed in their own `\plan` subdirectories (for example `\plan1` and `\plan2`), and the WebLogic Remote Console cannot identify them. Therefore, the `config.xml` must specify the plan you want to use. For information on `config.xml`, see Domain Configuration Files in *Understanding Domain Configuration for Oracle WebLogic Server*.

After you install a new application and existing deployment plan, the WebLogic Remote Console validates the deployment plan configuration against the target servers and clusters that were selected during installation. If the deployment plan contains empty (null) variables, or

if any values configured in the deployment plan are not valid for the target server instances, you must override the deployment plan before you can deploy the application. You can also configure tuning parameters to better suit the target environment in which you are deploying the application, as described in Saving Configuration Changes to a Deployment Plan. Changes you make to the application's configuration are saved to a new deployment plan.

If you have a valid deployment plan that fully configures an application for the environment in which you are deploying, you can use either the WebLogic Remote Console or the `weblogic.Deployer` utility to deploy an application with a deployment plan to use for deployment.

> **Note:**
>
> Any deployment plan you use with the `weblogic.Deployer` utility must be complete and valid for your target servers. `weblogic.PlanGenerator` does not allow you to set or override individual deployment properties when it creates a plan. To deploy a new application and existing deployment plan using `weblogic.Deployer`, see Deploying an Application with a Deployment Plan.

You can use a deployment plan to override the `context-root` element defined in the `application.xml` deployment descriptor. The following example shows how to use a `Plan.xml` file to change the `context-root` of an application, with the relevant sections in bold.

Example 3-2 Using a Deployment Plan to Change the context-root of an Application

```
<deployment-plan xmlns="http://xmlns.oracle.com/weblogic/deployment-plan">
<application-name>test</application-name>
<variable-definition>
  <variable>
   <name>myNewRoot</name>
   <value>DEF</value>
  </variable>
</variable-definition>
<module-override>
  <module-name>test</module-name>
  <module-type>ear</module-type>
  <module-descriptor external="false">
    <root-element>application</root-element>
    <uri>META-INF/application.xml</uri>
    <variable-assignment>
     <name>myNewRoot</name>
     <xpath>/application/module/web/[context-root="ABC"]/</xpath>
     <operation>replace</operation>
    </variable-assignment>
  </module-descriptor>
</module-override>
<config-root>D:\Cases\WLS\PlanXML\test</config-root>
</deployment-plan>
```

In Example 3-2:

- The `context-root` defined in the `application.xml` file is `ABC`.
- Using the deployment plan, the `context-root` successfully changes from `ABC` to `DEF`.
- The URL used to access the application changes from `http://localhost:7001/ABC/index.html` to `http://localhost:7001/DEF/index.html`.

Generic File Loading Overrides

Generic file loading overrides let you place application-specific files to be overridden into an optional subdirectory (named `/AppFileOverrides`) in the existing plan directory structure. The presence or absence of this subdirectory controls whether file overrides are enabled for the deployment.

If the subdirectory is present, an internal ClassFinder is added to the front of the application and module classloaders for the deployment. As a result, the file override hierarchy rules follow the existing classloader and resource loading rules and behaviors for applications. For more information on WebLogic Server application classloading, see WebLogic Server Application Classloading in *Developing Applications for Oracle WebLogic Server*.

> **Note:**
>
> This mechanism is for overriding resources only and does not override classes.

These are application-specific files and the contents are opaque to WebLogic Server, so the entire file contents will be overridden when an override file is supplied.

How It Works

The files placed in the `/AppFileOverrides` subdirectory are staged and distributed along with the rest of the plan directory contents and are available on all of the targets. Applications are then able to load these files as resources using the current classloader (for example, using the `ClassLoader.getResourceAsStream` method.) This either finds the overridden files or the files packaged in the application, depending on the configuration and whether overridden files are supplied.

For Web applications, application file overrides only apply to the classpath related resources (which are in `WEB-INF/classes` and `WEB-INF/lib`), and *do not apply* to the resource path for the Web application. Therefore, overrides are seen by Web applications using the `classloader.getResourceAsStream()` method to lookup resources, but overrides do not affect Web application calls to the `ServletContext.getResourceAsStream()` method.

In order to use this feature, you must:

- Specify a plan for the deployment (see Creating a New Deployment Plan to Configure an Application).
- Specify the `config-root` within in the plan.
- Provide a `config-root/AppFileOverrides` subdirectory.

Directory Structure

The contents of the `/AppFileOverrides` subdirectory use the existing plan directory structure and directory naming conventions that already exist for descriptor overrides. For more information on directory naming conventions, see Packaging Files for Deployment.

Enabling application file overrides causes a directory ClassFinder to be added to the application and module level classloaders, which point to the appropriate root directories within the `/AppFileOverrides` subdirectory (which is in the plan directory). The ClassFinder inserted into the front of the application's classloader is given a structure of `AppDeploymentMBean.getLocalPlanDir` + `separator` + `"/AppFileOverrides"`. The ClassFinder inserted into the front of the module's classloaders is given a structure of `AppDeploymentMBean.getLocalPlanDir` + `separator` + `"/AppFileOverrides"` + `separator` + `moduleURI`.

For example:

Table 3-1 Directory Structure for Generic File Overrides

Directory	Description
`install-root/plan/AppFileOverrides`	Directory put in front of the main application classloader's classpath
`install-root/plan/AppFileOverrides/WebApp1.war/...`	Directory put in front of the `WebApp1.war` classloader's classpath
`install-root/plan/AppFileOverrides/WebApp2.war/...`	Directory put in front of the `WebApp2.war` classloader's classpath

Application Usage

It is important to note that the application controls the file contents and format and controls whether the contents of the files are accessed by the application code. As a best practice, generic file loading overrides should be used by application code that has environment-specific properties files, and which is loading those properties files as resources using the application's classloader. For example, the application code may do the following:

```
Properties myAppProps = new Properties();
InputStream iostream =
Thread.currentThread().getContextClassLoader().getResourceAsStream("myCfg/
myApp.properties");
myAppProps.load(iostream);
```

Additional Configuration Tasks

The referenced information describes additional deployment configuration tasks.

- Deploying an Application with a Deployment Plan, describes how to deploy an application with a valid deployment plan using the `weblogic.Deployer` tool. See weblogic.Deployer Command-Line Reference.
- Updating the Deployment Configuration for an Application, describes how to update the deployment configuration for a currently-deployed application.
- Exporting an Application for Deployment to New Environments, explains how developers can create portable deployment plans using the `weblogic.PlanGenerator` tool.

- weblogic.PlanGenerator Command-Line Reference, provides a complete reference to the `weblogic.PlanGenerator` tool.

Best Practices for Managing Application Configuration

- Always manage multiple deployment configurations using deployment plans, rather than multiple versions of the WebLogic Server deployment descriptor files.
- Always store each existing deployment plan for an application in its own `plan` subdirectory of an application root directory.
- If your organization requires standardized, repeatable deployments to several environments, use the Application with Single Deployment Plan workflow to maintain a single deployment plan in your source control system.
- If you make extensive changes to an application's deployment configuration using the WebLogic Remote Console, back up or safely store the updated deployment plan for future use. Oracle recommends storing the entire application root directory in a source control system, so that you can maintain configuration information for multiple environments and multiple versions of an application.

4

Exporting an Application for Deployment to New Environments

Learn how to export an application's deployment configuration into a custom deployment plan. This process helps administrators easily deploy the application into non-development environments.
This chapter includes the following sections:

- Overview of the Export Process
  Exporting an application's deployment configuration is the process of creating a custom deployment plan for deploying the application into new WebLogic Server environments. When the process is complete, the application deployment files and the custom deployment plan are distributed to deployers (for example, testing, staging, or production administrators) who then use the deployment plan as a blueprint for configuring the application for their environment.
- Understanding Deployment Property Classifications
  Each WebLogic Server deployment descriptor property (for all Jakarta EE module descriptors as well as JDBC, JMS, and WLDF application modules) can be classified into one of four categories.
- Steps for Exporting an Application's Deployment Configuration
  Exporting an application's deployment configuration typically involves several steps.
- Staging Application Files for Export
  Oracle recommends placing application files into an application installation directory before exporting the deployment configuration.
- Generating a Template Deployment Plan using weblogic.PlanGenerator
  The `weblogic.PlanGenerator` tool provides a quick and easy way to generate a template deployment plan with null variables for an entire category (such as declaration or configurable properties) of deployment descriptors.
- Customizing the Deployment Plan Using the Remote Console
  A developer generally customizes the template plan to add one or more WebLogic Server tuning properties for the application.
- Manually Customizing the Deployment Plan
  In some cases you may need to edit a custom deployment plan manually, using a text editor.
- Best Practices for Exporting a Deployment Configuration
  Keep in mind recommended best practices when exporting an application's deployment configuration.

Overview of the Export Process

Exporting an application's deployment configuration is the process of creating a custom deployment plan for deploying the application into new WebLogic Server environments. When the process is complete, the application deployment files and the custom deployment plan are distributed to deployers (for example, testing, staging, or production administrators) who then use the deployment plan as a blueprint for configuring the application for their environment.

An administrator can install the application and the custom deployment plan using the WebLogic Remote Console, which validates the deployment plan and allows the administrator to update configuration properties needed for a specific deployment.

See the Understanding Deployment Plan Contents for more information about deployment plans.

- Goals for Exporting a Deployment Configuration
- Tools for Exporting a Deployment Configuration

Goals for Exporting a Deployment Configuration

The primary goals in exporting a deployment configuration are:

- **To expose the external resources requirements of the application as null variables in a deployment plan.** 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 may 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. Using empty (null) variables forces the deployer to fill in a valid resource name before the application can be deployed.
- **To expose additional configurable properties, such as tuning parameters, as variables in a deployment plan.** Certain tuning parameters that are acceptable in a development environment may be 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. Exporting selected tunables as deployment plan variables helps an administrator focus on important tuning parameters when deploying the application. The WebLogic Remote Console highlights tuning parameters exposed as variables in a deployment plan, but does not require a deployer to modify them before deployment.

Tools for Exporting a Deployment Configuration

WebLogic Server provides the following tools to help you export an application's deployment configuration:

- `weblogic.PlanGenerator` creates a template deployment plan with null variables for selected categories of WebLogic Server deployment descriptors. This tool is recommended if you are beginning the export process and you want to create a template deployment plan with null variables for an entire class of deployment descriptors (see Understanding Deployment Property Classifications). You typically need to manually modify the deployment plan created by `weblogic.PlanGenerator`, either manually or using the WebLogic Remote Console, to delete extraneous variable definitions or add variables for individual properties.
- The WebLogic Remote Console updates or creates new deployment plans as necessary when you change configuration properties for an installed application. You can use the WebLogic Remote Console to generate a new deployment plan or to add or override variables in an existing plan. The WebLogic Remote Console provides greater flexibility than `weblogic.PlanGenerator`, because it allows you to interactively add or edit individual deployment descriptor properties in the plan, rather than export entire categories of descriptor properties.

Understanding Deployment Property Classifications

Each WebLogic Server deployment descriptor property (for all Jakarta EE module descriptors as well as JDBC, JMS, and WLDF application modules) can be classified into one of four categories.

- *Non-configurable* properties cannot be changed by an administrator during a deployment configuration session. Non-configurable properties are used to describe application behavior that is fundamental to the basic operation of the application. For example, the `ejb-name` property is categorized as non-configurable, because changing its value also requires changing the EJB application code.
- *Dependency* properties resolve resource dependencies defined in the Jakarta EE deployment descriptors. For example, if the Jakarta EE descriptor for an EJB defines a datasource name that is used within the EJB code, the WebLogic Server descriptor uses a dependency property to bind the data source name to an actual data source configured in the target WebLogic Server domain.
- *Declaration* properties declare a resource that other applications can use. For example, the JNDI name of an EJB declares the EJB name that other applications or modules would use to access the EJB.
- *Configurable* properties are the remaining properties not classified as dependency or declaration properties. Generally configurable properties enable or configure WebLogic Server-specific features and tuning parameters for the deployed application. For example, the WebLogic Server descriptor for an EJB might define the number of EJBs that WebLogic Server caches in memory.

Use these categories during the configuration export process to select properties to expose as variables in the deployment plan. For example, a developer can generate a new deployment plan containing variable definitions for all properties tagged as "dependencies" in an application's WebLogic Server deployment descriptors. The variables can then be easily changed by an administrator deploying the application to an environment having different resource names.

All changeable descriptor properties (dependency, declaration, and configurable properties) are further classified as either *dynamic* or *non-dynamic* properties. Dynamic properties can be changed in a deployed application without requiring you to redeploy for the changes to take effect. Non-dynamic properties can be changed but require redeployment for the changes to take effect. The WebLogic Remote Console identifies non-dynamic properties as a reminder for when redeployment is necessary.

Steps for Exporting an Application's Deployment Configuration

Exporting an application's deployment configuration typically involves several steps.

The sections that follow describe each procedure in detail.

Staging Application Files for Export

Oracle recommends placing application files into an application installation directory before exporting the deployment configuration.

When using an installation directory, generated configuration files, such as the deployment plan, are automatically copied to the `\plan` subdirectory during export.

To create an application installation directory:

1. Create a top-level installation directory for your application:

   ```
   mkdir c:\exportapps\myApplication
   ```

2. Create `\app and \plan` subdirectories:

   ```
   mkdir c:\exportapps\myApplication\app
   mkdir c:\exportapps\myApplication\plan
   ```

3. Copy the complete application to be exported into the `\app` subdirectory. The application can be either in archive or exploded archive form:

   ```
   cp -r c:\dev\myApplication c:\exportapps\myApplication\app
   ```

   The `\app` directory must include the full application distribution, and can include the WebLogic Server descriptor files that you use for deployment to your development environment.

If you choose not to use an installation directory when exporting an application, Oracle recommends using the `-plan` option to `weblogic.PlanGenerator` to specify the location and filename of the generated plan. By default, `weblogic.PlanGenerator` stores generated files in the *`TEMP`*`/weblogic-install/`*`application_name`*`/config` directory, where *`TEMP`* is the temporary directory for your environment. For Windows platforms, this means generated configuration files are stored in `C:\Documents and Settings\username\Local Settings\`*`Temp`*`\weblogic\install\myApplication.ear\config`. Use the `-plan` option to place generated files in a known location.

Generating a Template Deployment Plan using weblogic.PlanGenerator

The `weblogic.PlanGenerator` tool provides a quick and easy way to generate a template deployment plan with null variables for an entire category (such as declaration or configurable properties) of deployment descriptors.

Oracle recommends using `weblogic.PlanGenerator` to generate a new deployment plan with null variables for all of an application's dependencies. This ensures that all global resources required for an application can be easily configured by administrators who must deploy the application in a new environment.

When using an application staged in an installation root directory, the basic syntax for using `weblogic.PlanGenerator` is:

```
java weblogic.PlanGenerator -root install_root category
```

where:

`install_root` specifies the fully qualified name of the root directory for the application and plan.

`category` specifies the category of WebLogic Server deployment descriptors for which you want to create variables. (See Understanding Deployment Property Classifications for a description of each category.) For the purposes of generating a template deployment plan, you should usually use only the `-dependencies` option, which is the default option, as this limits variables to external resources required by the application.

> **Note:**
>
> The `-dependencies` option creates null variables for every possible dynamically-configurable deployment property, which can result in a large number of variable definitions that may not be required for your application. The -declarations option is generally not required, because declaration properties are typically associated with the basic functioning of the application and should not be changed before deployment to a new environment.

For example:

```
java weblogic.PlanGenerator -root c:\exportapps\myApplication -dependencies

java weblogic.PlanGenerator -root c:\exportapps\myApplication
```

With the previous commands, which are synonymous because `-dependencies` is the default option so you are not required to specify it in your `weblogic.PlanGenerator` command, `weblogic.PlanGenerator` inspects all Jakarta EE deployment descriptors in the selected application, and creates a deployment plan with null variables for all relevant WebLogic Server deployment properties that configure external resources for the application. Using this template deployment plan, an administrator using the WebLogic Remote Console is directed to assign valid resource names and tuning properties for each null variable before the application can be deployed.

Customizing the Deployment Plan Using the Remote Console

A developer generally customizes the template plan to add one or more WebLogic Server tuning properties for the application.

The template deployment plan generated in Generating a Template Deployment Plan using weblogic.PlanGenerator contains only those deployment properties that resolve external dependencies for the application. The WebLogic Remote Console enables you to easily add deployment plan variables for individual deployment descriptor properties as needed. To customize a deployment plan using the WebLogic Remote Console, follow the steps in the following topics:

- Install the Exported Application and Template Deployment Plan
- Add Variables for Selected Tuning Properties
- Retrieve the Customized Deployment Plan

Install the Exported Application and Template Deployment Plan

To modify a deployment configuration using the WebLogic Remote Console, you must first install the application and existing deployment plan as described in Steps for Creating an Application Installation Directory.

Add Variables for Selected Tuning Properties

After installing the exported application, follow the steps in Specify Deployment Properties in the *Oracle WebLogic Remote Console Online Help* to add new tuning properties to the deployment plan.

Retrieve the Customized Deployment Plan

When you modify an application's deployment configuration using the WebLogic Remote Console, your changes to deployment properties are stored in a WebLogic Server deployment plan and/or in generated WebLogic Server deployment descriptor files. If you modify any deployment properties defined as variables in the application's deployment plan, your changes are written back to a new version of the plan file. If the application that was installed from an installation directory, the WebLogic Remote Console stores the generated configuration files in the `plan` subdirectory by default.

Manually Customizing the Deployment Plan

In some cases you may need to edit a custom deployment plan manually, using a text editor.

This may be necessary for the following reasons:

- You want to remove an existing deployment plan variable.
- You want to assign a null value to a generated variable in the plan.

> **Note:**
>
> You cannot use the WebLogic Remote Console to remove variable definitions from the deployment plan or assign a null value for a deployment property.

See `http://xmlns.oracle.com/weblogic/deployment-plan/1.01/deployment-plan.xsd` and Understanding Deployment Plan Contents before manually editing deployment plan entries.

- Removing Variables from a Deployment Plan
- Assigning Null Variables to Require Administrator Input

Removing Variables from a Deployment Plan

The `variable-definition` stanza in a deployment plan defines the names and values of variables used for overriding WebLogic Server deployment descriptor properties. The `module-override` element may contain one or more `variable-assignment` elements that define where a variable is applied to a given deployment descriptor. To remove a variable from a deployment plan, use a text editor to delete:

- The `variable` definition from the `variable-definition` stanza
- All `variable-assignment` elements that reference the deleted variable.

Assigning Null Variables to Require Administrator Input

To assign a null value to an existing variable definition, simply change any text value that is present in the `value` subelement in the `variable` element to `<value xsi:nil="true"></value>` where the xsi namespace is defined as: `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`. For example, change:

```
...
  <variable-definition>
    <variable>
```

```
      <name>SessionDescriptor_InvalidationIntervalSecs_11029744771850</name>
      <value>80</value>
    </variable>
  </variable-definition>
...
```

to:

```
...
  <variable-definition>
    <variable>
      <name>SessionDescriptor_InvalidationIntervalSecs_11029744771850</name>
      <value xsi:nil="true"></value>
    </variable>
  </variable-definition>
...
```

Best Practices for Exporting a Deployment Configuration

Keep in mind recommended best practices when exporting an application's deployment configuration.

- The primary goal for exporting an application is to create null variables for all of an application's external resource dependencies. This ensures that deployers have the ability to assign resource names based on resources available in their target environment.
- Use `weblogic.PlanGenerator` only for exporting resource dependencies. Using `weblogic.PlanGenerator` to export other categories of deployment descriptor properties generally results in too many variables in the deployment plan.
- Use the WebLogic Remote Console to add individual tuning property values to the deployment plan, or to validate a custom deployment plan.
- Neither the WebLogic Remote Console nor `weblogic.PlanGenerator` allow you to remove variables from a plan or set null values for variables. Use a text editor when necessary to complete these tasks.

5

Deploying Applications and Modules with weblogic.Deployer

Learn how WebLogic Server administrators and developers perform interactive, command-line based deployment tasks using the `weblogic.Deployer` utility.
This chapter includes the following sections:

- Overview of Common Deployment Scenarios
  Learn these common deployment tasks which are organized into several general categories, starting with the most basic tasks and progressing to more advanced tasks.
- Uploading Deployment Files from a Remote Client
  In order to deploy an application or module to a domain, the deployment file(s) must be accessible to the domain's Administration Server. If the files do not reside on the Administration Server machine or are not available to the Administration Server machine via a network mounted directory, use the `-upload` option to upload the files before deploying them.
- Deploying to a Single-Server Domain
  A single-server WebLogic Server domain, consisting only of an Administration Server, represents the simplest scenario in which to deploy an application or module. If you are deploying files that reside on the same machine as the domain, use the **`-deploy`** command and identify the file location, with connection arguments for the Administration Server.
- Deploying an Application with a Deployment Plan
  When you use `weblogic.Deployer` to deploy an application, the deployment plan and WebLogic Server deployment descriptors must define a valid configuration for the target environment, or the deployment fails. This means you cannot use `weblogic.Deployer` with a deployment plan that defines null variables for an application's required resource bindings.
- Deploying an Application That Looks Up System Resources from JNDI During preStart
  Because preStart application life cycle listeners are invoked during the prepare phase of an application—which occurs when an edit session is committed—JNDI lookup of system resources fails if you create a new system resource during the same edit session in which you deploy an application that uses the system resource.
- Targeting Deployments to Servers, Clusters, and Virtual Hosts
  In most production environments, you typically deploy applications to one or more Managed Servers configured in a WebLogic Server domain.
- Using Module-Level Targeting for Deploying an Enterprise Application
  An enterprise application (EAR file) differs from other deployment units because an EAR can contain other module types (WAR and JAR archives).
- Deploying JDBC, JMS, and WLDF Application Modules
  Standalone JDBC, JMS, and WLDF application modules can be deployed similar to standalone Jakarta EE modules. For a standalone JDBC, JMS, or WLDF application module, the target list determines the WebLogic Server domain in which the module is available.

- Controlling Deployment File Copying with Staging Modes
  The deployment *staging mode* determines how deployment files are made available to target servers that must deploy an application or standalone module.
- Distributing Applications to a Production Environment
  Distributing an application prepares it for deployment by copying its deployment files to all target servers and validating it. After you distribute an application, you can start it in administration mode, which restricts access to the application to a configured administration channel and allows you to distribute the application to a production environment (or distribute a new version of an application) without opening the application to external client connections.
- Deploying Shared Jakarta EE Libraries and Dependent Applications
  Jakarta EE library support in WebLogic Server provides an easy way to share one or more Jakarta EE modules or JAR files among multiple enterprise applications.
- Best Practices for Deploying Applications
  Follow Oracle-recommended best practices when deploying applications.

Overview of Common Deployment Scenarios

Learn these common deployment tasks which are organized into several general categories, starting with the most basic tasks and progressing to more advanced tasks.

Before you deploy an application, perform the appropriate tasks described in Configuring Applications for Production Deployment.

- Uploading Deployment Files from a Remote Client—Explains how to upload an application or module to the Administration Server from a remote client.
- Deploying to a Single-Server Domain—Explains the basics of deploying an application or module to a single-server WebLogic domain.
- Deploying an Application with a Deployment Plan—Explains how to deploy an application with a deployment plan that fully configures the application.
- Deploying an Application That Looks Up System Resources from JNDI During preStart—Explains how to deploy an application that looks up system resources via JNDI.
- Targeting Deployments to Servers, Clusters, and Virtual Hosts—Describes all WebLogic Server target types, and explains how to identify targets during deployment.
- Using Module-Level Targeting for Deploying an Enterprise Application—Describes how to target individual modules in an enterprise application to different WebLogic Server targets.
- Deploying JDBC, JMS, and WLDF Application Modules—Describes how to deploy both standalone and application-scoped JDBC, JMS, and WLDF modules in a WebLogic Server domain.
- Controlling Deployment File Copying with Staging Modes—Describes how to use non-default staging modes to control WebLogic Server file-copying behavior.
- Distributing Applications to a Production Environment—Describes how to deploy and test a new application directly in a production environment *without* making the application available for processing client requests.
- Deploying Shared Jakarta EE Libraries and Dependent Applications—Describes how to deploy Jakarta EE Libraries and optional packages to a WebLogic Server environment, and how to deploy and manage applications and modules that reference these shared resources.
- Best Practices for Deploying Applications—Summarizes key deployment practices.

Uploading Deployment Files from a Remote Client

In order to deploy an application or module to a domain, the deployment file(s) must be accessible to the domain's Administration Server. If the files do not reside on the Administration Server machine or are not available to the Administration Server machine via a network mounted directory, use the `-upload` option to upload the files before deploying them.

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -deploy -upload c:\localfiles\myapp.ear
```

To upload an exploded archive directory, specify the directory name instead of an archive filename (for example `c:\localfiles\myappEar`).

When you upload files to the Administration Server machine, the archive file is automatically placed in the server's upload directory. You can configure the path of this directory using the instructions in Changing the Default Staging Behavior for a Server.

Upload Behavior When Updating an Application or Plan

When mixing local and remote (upload) deployment operations, it is very important to use a process that tracks how an application is deployed. Consider the following series of events:

1. An administrator or developer deploys an app (`myapp.ear`) with a deployment plan (`productionEnvPlan.xml`) to an Administration Server having a Managed Server using:

   ```
   java weblogic.Deployer -adminurl http://test:7001 -username weblogic
      -password weblogic -deploy c:\localfiles\myapp.ear
      -plan c:\localfiles\productionEnvPlan.xml
   ```

2. If the administrator or developer uses WebLogic Portal on the Managed Server and saves some configuration changes to the `myapp.ear` application. When the application is saved, WebLogic Portal updates the application using the following:

   ```
   java weblogic.Deployer -adminurl http://test:7001 -username weblogic
      -password weblogic -update -name myapp.ear -upload
      -plan c:\localfiles\nuPlan.xml
   ```

3. The application and deployment plan file from the remote machine are uploaded to the Administration Server upload directory. The deployment is updated and the configuration uses the deployment plan at `c:\domain\servers\adminServerName\upload\plan\nuPlan.xml`.

At this point, an administrator or developer expecting the behavior derived from the application (`c:\localfiles\myapp.ear`) and deployment plan (`c:\localfiles\productionEnvPlan.xml`) can easily become confused. If the administrator or developer:

- Reviews the application and deployment plan files, the files have not changed. New behaviors appear to be unexplained.
- Makes changes to the `c:\localfiles\myapp.ear` application file or `c:\localfiles\productionEnvPlan.xml` deployment plan file `xml`, the deployment configuration changes in the upload directory are at risk of being lost.

Deploying to a Single-Server Domain

A single-server WebLogic Server domain, consisting only of an Administration Server, represents the simplest scenario in which to deploy an application or module. If you are deploying files that reside on the same machine as the domain, use the **-deploy** command and identify the file location, with connection arguments for the Administration Server.

For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -deploy c:\localfiles\myapp.ear
```

In the above command, WebLogic Server creates a default deployment name of `myapp`, as described in Understanding Default Deployment Names, because `myapp` is the name of the deployment file without the extension. If you want to specify a non-default deployment name, use the `-name` option, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001
   -username weblogic -password weblogic -deploy
   -name myTestApplication c:\localfiles\myapp.ear
```

To deploy an application or module to multiple targets, see Targeting Deployments to Servers, Clusters, and Virtual Hosts.

Deploying an Application with a Deployment Plan

When you use `weblogic.Deployer` to deploy an application, the deployment plan and WebLogic Server deployment descriptors must define a valid configuration for the target environment, or the deployment fails. This means you cannot use `weblogic.Deployer` with a deployment plan that defines null variables for an application's required resource bindings.

To deploy an application and deployment plan using `weblogic.Deployer`, include the `-plan` option with the **-deploy** command, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
     -password weblogic -deploy -name myTestDeployment
     -source /myDeployments/myApplication.ear
     -targets myCluster -stage
     -plan /myDeployments/myAppPlan.xml
```

If you are deploying from an application root directory and the deployment plan is located in the `/plan` subdirectory, you still need to identify both the actual deployment source files the plan to use for deployment, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
     -password weblogic -deploy -name myTestDeployment
     -source /myDeployments/installedApps/myApplication/app/myApplication.ear
     -targets myCluster -stage
     -plan /myDeployments/installedApps/myApplication/plan/plan.xml
```

When you deploy or distribute an application with a deployment plan, the deployment plan and any generated deployment descriptors are copied to the staging directories of target servers along with the application source files.

You can also stage deployment plans independently of the application archive. See Staging Deployment Plans.

Deploying an Application That Looks Up System Resources from JNDI During preStart

Because preStart application life cycle listeners are invoked during the prepare phase of an application—which occurs when an edit session is committed—JNDI lookup of system resources fails if you create a new system resource during the same edit session in which you deploy an application that uses the system resource.

For example, if you create a new JDBC system resource in the WebLogic Remote Console and then deploy an application that uses the new JDBC system resource during the same edit session (in other words, before committing the changes in the WebLogic Remote Console), the JNDI lookup of the JDBC system resource will fail because the preStart application life cycle listeners were invoked before the system resource was activated. To look up system resources from JNDI in your preStart life cycle listeners, Oracle recommends that you create the system resource during startup or during a separate edit session, before you deploy any applications that use the system resource.

For example, using the WebLogic Remote Console:

1. In the **Edit Tree**, create the desired system resource. See Create a JMS System Module in the *Oracle WebLogic Remote Console Online Help*.
2. To complete the edit session, open the Shopping Cart and then click **Commit Changes**.
3. Again, using the **Edit Tree**, deploy the application that uses the system resource you created in step 1. For information on deploying applications from the WebLogic Remote Console, see Deploying Applications in the *Oracle WebLogic Remote Console Online Help*.

For more information, see Overview of Jakarta EE Libraries and Optional Packages in *Developing Applications for Oracle WebLogic Server*.

Targeting Deployments to Servers, Clusters, and Virtual Hosts

In most production environments, you typically deploy applications to one or more Managed Servers configured in a WebLogic Server domain.

In some cases, the servers may be included as part of a WebLogic Server cluster, or a virtual host may be used for directing Web application requests. The term *deployment target* refers to any server or collection of servers to which you can deploy an application or module.

Understanding Deployment Targets

Deployment targets are the servers or groups of servers to which you deploy an application or standalone module. During the deployment process, you select the list of targets from the available targets configured in your domain. You can also change the target list after you have deployed a module.

The following table describes all valid deployment targets for WebLogic Server, and lists the types of modules that you can deploy to each target.

Table 5-1 WebLogic Server Deployment Targets

Target Type	Description	Valid Deployments
WebLogic Server Instance	A WebLogic Server instance, such as an Administration Server in a single-server domain, or a Managed Server.	Jakarta EE applications Jakarta EE modules JMS, JDBC, or WLDF modules Jakarta EE libraries
Cluster	A configured cluster of multiple WebLogic Server instances	Jakarta EE applications Jakarta EE modules JMS, JDBC, or WLDF modules Jakarta EE libraries
Virtual Host	A configured host name that routes requests for a particular DNS name to a WebLogic Server instance or cluster. See Configuring Virtual Hosting in *Administering Server Environments for Oracle WebLogic Server* for more information.	Web applications
JMS Server	A JMS Server configured in a WebLogic Server domain	A JMS queue or topic defined within a JMS module* *When deployed as a standalone application module, a JMS, JDBC, or WLDF resource appears as a Jakarta EE deployment in the WebLogic Remote Console. A standalone JMS application module can be targeted to server, cluster, or virtual host targets; queues and topics defined within a JMS module can be further targeted to a configured JMS server. For information on sub-module targeting, see Using Sub-Module Targeting with JMS Application Modules.

Deploying to One or More Targets

To deploy to a single WebLogic Server target, list the configured target name after the `-targets` option to `weblogic.Deployer`. For example, to deploy a Web application to a configured virtual host named `companyHost`, use the command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -deploy -targets companyHost c:\localfiles\myWebApp.ear
```

Specify multiple targets using a comma-separated list, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -deploy -targets ManagedServer-1,ManagedServer-2
   c:\localfiles\myapp.ear
```

Deploying to a Cluster Target

If you specify a cluster target (using `-targets mycluster`), WebLogic Server targets all server instances in the cluster by default. This corresponds to homogenous module deployment, which is recommended in most clusters. If you want to deploy a module only to a single server in the cluster (that is, "pin" a module to a server), specify the individual server instance name, rather than the cluster, as the target. This type of deployment is less common, and should be

used only in special circumstances where pinned services are required. See Understanding Cluster Configuration and Application Deployment in *Administering Clusters for Oracle WebLogic Server* for more information.

> **Note:**
>
> Pinning a deployment to a subset of server instances in a cluster (rather than to a single server on the cluster) is not recommended and will generate a warning message.

When you deploy an application to a cluster target, WebLogic Server ensures that the deployment successfully deploys on all available members of the cluster. If even a single, available WebLogic Server instance in the cluster cannot deploy the application, the entire deployment fails and no servers in the cluster start the application. This helps to maintain homogeneous deployments to the cluster, because deployment operations succeed or fail as a logical unit.

If a clustered server is unreachable at the time of deployment (for example, because of a network failure between the Administration Server and a Managed Server, or because a cluster member is shut down) that server does not receive the deployment request until the network connection is restored. This default behavior ensures that most deployment operations succeed, even when servers are taken offline.

Enforcing Consistent Deployment to All Configured Cluster Members

The default cluster deployment behavior ensures homogeneous deployment for all clustered server instances that can be reached at the time of deployment. However, if the Administration Server cannot reach one or more clustered servers due to a network outage, those servers do not receive the deployment request until the network connection is restored. For redeployment operations, this can lead to a situation where unreachable servers use an older version of the deployed application, while reachable servers use the newer version. When the network connection is restored, previously-disconnected servers may abruptly update the application as they receive the delayed redeployment request.

It is possible to change WebLogic Server's default deployment behavior for clusters by setting the `ClusterConstraintsEnabled` option when starting the WebLogic Server domain. The `ClusterConstraintsEnabled` option enforces strict deployment for all servers configured in a cluster. A deployment to a cluster succeeds only if all members of the cluster are reachable and all can deploy the specified files.

> **Note:**
>
> Do not use the `ClusterConstraintsEnabled` option unless you have an extremely reliable network configuration, and you can guarantee that all cluster members are always available to receive deployment and redeployment requests. With `ClusterConstraintsEnabled`, WebLogic Server will fail all deployment operations to a cluster if any clustered server is unavailable, even if a single server has been shut down for maintenance.

To set the `ClusterConstraintsEnabled` for the domain when you start the Administration Server, include the appropriate startup argument:

- `-DClusterConstraintsEnabled=true` enforces strict cluster deployment for servers in a domain.
- `-DClusterConstraintsEnabled=false` ensures that all available cluster members deploy the application or module. Unavailable servers do not prevent successful deployment to the available clustered instances. This corresponds to the default WebLogic Server deployment behavior.

Using Module-Level Targeting for Deploying an Enterprise Application

An enterprise application (EAR file) differs from other deployment units because an EAR can contain other module types (WAR and JAR archives).

When you deploy an enterprise application using weblogic.Deployer, you can target all of the archive's modules together as a single deployment unit, or you can use module-level targeting to deploy only a subset of the modules available in an EAR. This can simplify packaging and distribution of applications by packaging multiple modules in a single, distributable EAR, but targeting only the modules you need to each domain.

Module-Targeting Syntax

To target individual modules in an enterprise application, use the *module_name*@*target_name* syntax. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name myEnterpriseApp
   -targets module1@myserver1,module2@myserver2,module3@myserver3
   -stage -deploy c:\localfiles\myEnterpriseApp.ear
```

Targeting Web Application Modules

To target Web application modules that are part of an `.ear` file, you can use the Web application's `context-root` name as the module name or specify the `web-uri`.

For example, if the `application.xml` file for a file, `myEnterpriseApp.ear`, defines:

```
<module>
  <web>
    <web-uri>myweb.war</web-uri>
    <context-root>/welcome</context-root>
  </web>
</module>
```

You can deploy only the Web application module by using the `context-root` name:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name mywebapplication -targets /welcome@myserver1
   -stage -deploy c:\localfiles\myEnterpriseApp.ear
```

You can deploy only the Web application module by using the `web-uri`:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name mywebapplication -targets myweb.war@myserver1
   -stage -deploy c:\localfiles\myEnterpriseApp.ear
```

To deploy a Web application as a default Web application, set the value of the context-root element to "/". For example, if the `application.xml` file for a file, `myEnterpriseApp.ear`, defines:

```
<module>
  <web>
    <web-uri>myweb.war</web-uri>
    <context-root>/</context-root>
  </web>
</module>
```

You can deploy only the Web application module by using the `context-root` name:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name mywebapplication -targets /@myserver1
   -stage -deploy c:\localfiles\myEnterpriseApp.ear
```

Starting and Stopping Modules

You can start or stop individual child modules, as follows:

```
java weblogic.Deployer -username weblogic -password weblogic -adminurl
t3://localhost:7001 -name appName -start -targets moduls1@myserver1

java weblogic.Deployer -username weblogic -password weblogic -adminurl
t3://localhost:7001 -name appName -stop -targets moduls1@myserver1
```

> **Note:**
>
> Use caution starting and stopping modules in production when modules have dependencies.

Deploying JDBC, JMS, and WLDF Application Modules

Standalone JDBC, JMS, and WLDF application modules can be deployed similar to standalone Jakarta EE modules. For a standalone JDBC, JMS, or WLDF application module, the target list determines the WebLogic Server domain in which the module is available.

JNDI names specified within an application module are bound as global names and available to clients. For example, if you deploy a standalone JDBC application module to a single-server target, then applications that require resources defined in the JDBC module can only be deployed to the same server instance. You can target application modules to multiple servers, or to WebLogic Server clusters to make the resources available on additional servers.

If you require JDBC, JMS, or WLDF resources to be available to all servers in a domain, create system modules, rather than deployable application modules. See Configuring JMS System Resources in *Administering JMS Resources for Oracle WebLogic Server*, Configuring WebLogic JDBC Resources in *Administering JDBC Data Sources for Oracle WebLogic Server*, and Configuring Diagnostic System Modules in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

> **Note:**
>
> Deploying a JMS module does not necessarily mean it is properly targeted and active, and that the JNDI name is available. The JNDI name is not available until the JMS module is properly targeted. See Understanding JMS Resource Configuration in *Administering JMS Resources for Oracle WebLogic Server*.

- Targeting Application-Scoped JMS, JDBC, and WLDF Modules
- Using Submodule Targeting with JMS Application Modules
- Deploying Oracle Wallet Files for JDBC Modules

Targeting Application-Scoped JMS, JDBC, and WLDF Modules

JMS, JDBC, and WLDF application modules can also be included as part of an enterprise application, as an application-scoped resource module. Application-scoped resource modules can be targeted independently of other EAR modules during deployment, if necessary, by using module-level targeting syntax. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name myEnterpriseApp
   -targets myWebApp@myCluster,myJDBCModule@myserver1,myEJBModule@myserver1
   -stage -deploy c:\localfiles\myEnterpriseApp.ear
```

Using Submodule Targeting with JMS Application Modules

Certain JMS resources defined within a JMS application module can be further targeted to a JMS Server available on the module's target during deployment. These resources are known as submodules. Certain types of submodule require deployment to a JMS Server, such as:

- Queues
- Topics

Other submodules can be targeted to JMS Servers as well as WebLogic Server instances and clusters:

- Connection factories
- Foreign servers
- SAF imported destinations
- Uniform distributed topics
- Uniform distributed queues

To specify submodule targets at deployment or undeployment time, you must use an extended form of the module targeting syntax with the `-submoduletargets` option to `weblogic.Deployer`.

- Default Submodule Targeting
- Submodule Targeting for Standalone JMS Modules
- Submodule Targeting for Application-Scoped JMS Modules

Default Submodule Targeting

When deploying from the WebLogic Remote Console, WebLogic Server selects default JMS Server targets for submodules in a JMS application module, as described in Targeting JMS Modules and Subdeployment Resources in *Administering JMS Resources for Oracle WebLogic Server*.

When deploying submodule using WLST, you can use the parent module's targets as default targets for JMS resources when all the following conditions are met:

- The application contains one or more JMS mdoules.
- There is exactly one JMS server instance associated with the module target.
- The `-submoduletargets` option is not specified
- The defaultSubmoduleTargets option is true.

See Deployment Commands in *WebLogic Scripting Tool*.

Submodule Targeting for Standalone JMS Modules

For a standalone JMS module, the submodule targeting syntax is: `-submoduletargets` *`submodule_name@target_name`*. For example, to deploy a standalone JMS module on a single server instance, targeting the submodule `myQueue` to a JMS Server named JMSServer1, enter the command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name myJMSModule
   -targets ManagedServer1 -submoduletargets myQueue@JMSServer1
   -deploy c:\localfiles\myJMSModule.xml
```

To undeploy the same JMS module, enter the following command, which, assuming `myJMSModule` contains more than one submodule, will undeploy only the `myQueue` submodule; all other submodules are unaffected.

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name myJMSModule
   -undeploy -submoduletargets myQueue@JMSServer1
```

Submodule Targeting for Application-Scoped JMS Modules

For an application-scoped JMS module in an EAR, use the syntax: *`submodule_name@module_name@target_name`* to target a submodule. For example, if the queue in the above example were instead packaged as part of an enterprise application, you would use the command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name myEnterpriseApp
   -targets ManagedServer1 -submoduletargets myQueue@myJMSModule@JMSServer1
   -deploy c:\localfiles\myEnterpriseApp.ear
```

To undeploy the same JMS module, enter the following command, which, assuming `myEnterpriseApp` contains more than one submodule, will undeploy only the `myQueue` submodule; all other submodules are unaffected.

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name myEnterpriseApp
   -undeploy -submoduletargets myQueue@myJMSModule@JMSServer1
```

> **Note:**
>
> When you undeploy an application that contains application-scoped resources, the resources are deleted along with the application, which can potentially cause abandoned transactions or lost messages as a result of deleted JMS destinations. See Unregister Resource Grace Period in *Developing JTA Applications for Oracle WebLogic Server*.
>
> You should only undeploy applications that you are certain you want to completely remove; to temporarily stop client access to applications, use the `-stop` command, described in weblogic.Deployer Command-Line Reference.

For more information on JMS subdeployments, see Targeting JMS Modules and Subdeployment Resources in *Administering JMS Resources for Oracle WebLogic Server*.

Deploying Oracle Wallet Files for JDBC Modules

Oracle wallet files provide an easy method to manage database credentials across multiple domains.

You can update database credentials by updating the wallet instead of changing data source definitions. Similarly, you can use an Oracle TNS (Transparent Network Substrate) administrative file (`tnsnames.ora`) to hide the details of the database connection string (host name, port number, and service name) from the data source definition. If the connection information changes, simply change the `tnsnames.ora` file. For more information, see Creating and Managing Oracle Wallet in *Administering JDBC Data Sources for Oracle WebLogic Server*.

Oracle wallet files can be deployed as DBClientData modules. DBClientData modules may contain `tnsnames.ora` files, wallet files, server trust keystores, client identity keystores, basically all the database client connection data used by a data source, collocated in a new type of deployment module. DBClientData modules are standalone deployment modules that are independent of the configuration of the data source instances that are going to use those data. DBClientData modules make it easy to deploy, manage, and move database client data. For more information, see Use DBClientData Modules for Portability in *Administering JDBC Data Sources for Oracle WebLogic Server*.

You can manage DBClientData modules using existing WebLogic deployment tools, such as weblogic.Deployer, WLST online, REST APIs, WebLogic Remote Console, as well as WebLogic Deploy Tooling (WDT). The source of DBClientData modules can be a ZIP file or an exploded archive directory that contains the `tnsnames.ora` and wallet files. For detailed information, see Manage DBClientData Modules in *Administering JDBC Data Sources for Oracle WebLogic Server*.

The following examples illustrate how to deploy, distribute, undeploy, and redeloy DBClientData modules using weblogic.Deployer.

- Deploy DBClientData Modules
- Distribute DBClientData Modules
- Undeploy DBClientData Modules
- Redeploy DBClientData Modules

Deploy DBClientData Modules

The following example uses weblogic.Deployer to upload and deploy a DBClientData module.

```
java weblogic.Deployer
-name myNewApp3
-adminurl http://joesmith-1.subnet1ad3phx.devbox.companyvcn.com:9001
-username xxxxxx -password xxxxxx
-deploy -dbclientdata
-upload
/Users/joesmith/Downloads/myWallet.zip
```

Distribute DBClientData Modules

The following example uses weblogic.Deployer to propagate a DBClient Data module to Managed Servers when the files are already available on the Administration Server.

```
java weblogic.Deployer
-name myNewApp3
-adminurl http://joesmith-1.subnet1ad3phx.devbox.companyvcn.com:9001
-username xxxxxx -password xxxxxxxx
-distribute -dbclientdata
-remote
/scratch/joesmith/weblogic/dev/sandbox/joesmith/mydomain/config/dbclientdata/myNewApp3/
myWallet.zip
```

The following example uses weblogic.Deployer to upload a DBClient Data module to the Administration Server and propagate it to Managed Servers.

```
java weblogic.Deployer
-name myNewApp3
-adminurl http://joesmith-1.subnet1ad3phx.devbox.companyvcn.com:9001
-username xxxxxx -password xxxxxxxx
-distribute -dbclientdata
-upload
/Users/joesmith/Downloads/myWallet.zip
```

Undeploy DBClientData Modules

The following example uses weblogic.Deployer to undeploy a DBClientData module.

```
java weblogic.Deployer
-name myNewApp3
-adminurl http://joesmith-1.subnet1ad3phx.devbox.companyvcn.com:9001
-username xxxxxx -password xxxxxxxx
-undeploy
```

Redeploy DBClientData Modules

DBClientData modules are not versioned, therefore a redeploy operation is the same as an undeploy operation followed by a deploy operation.

The following example uses weblogic.Deployer to redeploy a DBClientData module.

```
-name myNewApp3
-adminurl http://joesmith-1.subnet1ad3phx.devbox.companyvcn.com:9001
-username xxxxxx -password xxxxxxxx
```

```
-redeploy -dbclientdata
-upload
/Users/joesmith/Downloads/myWallet2.zip
```

Controlling Deployment File Copying with Staging Modes

The deployment *staging mode* determines how deployment files are made available to target servers that must deploy an application or standalone module.

WebLogic Server provides three different options for staging files: stage mode, nostage mode, and external_stage mode.

Staging Mode Descriptions and Best Practices

The following table describes the behavior and best practices for using the different deployment staging modes.

Table 5-2 Application Deployment Staging Modes

Deployment Staging Mode	Behavior	When to Use
stage	The Administration Server first copies the deployment unit source files to the staging directories of target servers. (The staging directory is named `stage` by default, and it resides under the target server's root directory.) The target servers then deploy using their local copy of the deployment files.	• Deploying small or moderate-sized applications to multiple WebLogic Server instances. • Deploying small or moderate-sized applications to a cluster.
nostage	The Administration Server does not copy deployment unit files. Instead, all servers deploy using the same physical copy of the deployment files, which must be directly accessible by the Administration Server and target servers. With nostage deployments of exploded archive directories, WebLogic Server automatically detects changes to a deployment's JSPs or Servlets and refreshes the deployment. (This behavior can be disabled if necessary.)	• Deploying to a single-server domain. • Deploying to a cluster on a multi-homed machine. • Deploying very large applications to multiple targets or to a cluster where deployment files are available on a shared directory. • Deploying exploded archive directories that you want to periodically redeploy after changing content. • Deployments that require dynamic update of selected Deployment Descriptors via the WebLogic Remote Console.

Table 5-2 (Cont.) Application Deployment Staging Modes

Deployment Staging Mode	Behavior	When to Use
external_stage	The Administration Server does not copy deployment files. Instead, the administrator must ensure that deployment files are distributed to the correct staging directory location before deployment (for example, by manually copying files prior to deployment). With external_stage deployments, the Administration Server requires a copy of the deployment files for validation purposes. Copies of the deployment files that reside in target servers' staging directories are not validated before deployment. You can use the `-noversion` option to turn off the requirement that deployment files be on the Administration Server, but the `-noversion` option causes versioning information to be ignored; therefore, you cannot use the `-noversion` option with versioned applications. See Common Arguments.	• Deployments where you want to manually control the distribution of deployment files to target servers. • Deploying to domains where third-party applications or scripts manage the copying of deployment files to the correct staging directories. • Deployments that do not require dynamic update of selected Deployment Descriptors via the WebLogic Remote Console (not supported in external_stage mode). • Deployments that do not require partial redeployment of application components.

A server's staging directory is the directory in which the Administration Server copies deployment files for stage mode deployments. It is also the directory in which deployment files must reside before deploying an application using external_stage mode.

Most deployments use either stage or nostage modes, and the WebLogic Server deployment tools use the appropriate default mode when you deploy an application or module. For example, when deploying applications in WebLogic Server using weblogic.Deployer or WLST, if you do not specify a stage mode, the default stage mode is used; on the Administration Server, the default stage mode is nostage and on Managed Servers, it is stage.

The sections that follow explain how to explicitly set the deployment staging mode when deploying an application or module.

Using Nostage Mode Deployment

In nostage mode, the Administration Server does not copy the archive files from their source location. Instead, each target server must access the archive files from a single source directory for deployment. The staging directory of target servers is ignored for nostage deployments.

For example, if you deploy a Jakarta EE application to three servers in a cluster, each server must be able to access the same application archive files (from a shared or network-mounted directory) to deploy the application.

> **Note:**
>
> The source for the deployment files in nostage mode is the path provided by the user at deployment time (as opposed to stage mode, where the source is the path in each server's staging directory). However, even in nostage mode, WebLogic Server copies out parts of the deployment to temporary directories. This enables users to update entire archived deployments or parts of archived deployments.

In nostage mode, the Web application container automatically detects changes to JSPs and servlets. Nostage also allows you to later update only parts of an application by updating those parts in one file system location and then redeploying.

The WebLogic Remote Console uses nostage mode as the default when deploying only to the Administration Server (for example, in a single-server domain). `weblogic.Deployer` uses the target server's staging mode, and Administration Servers use nostage mode by default. You can also select nostage mode if you run a cluster of server instances on the same machine, or if you are deploying very large applications to multiple machines that have access to a shared directory. Deploying very large applications in nostage mode saves time during deployment because no files are copied.

Syntax for Nostage Mode

To use nostage mode, specify `-nostage` as an option to `weblogic.Deployer`, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name mydeploymentname
   -targets myserver1,myserver2,myserver3 -nostage
   -deploy c:\localfiles\myapp.ear
```

Using Stage Mode Deployment

In stage mode, the Administration Server copies the deployment files from their original location on the Administration Server machine to the staging directories of each target server. For example, if you deploy a Jakarta EE application to three servers in a cluster using stage mode, the Administration Server copies the deployment files to directories on each of the three server machines. Each server then deploys the Jakarta EE application using its local copy of the archive files.

When copying files to the staging directory, the Administration Server creates a subdirectory with the same name as the deployment name. So if you deployed using the command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name mytestear -stage -targets mycluster


   -deploy
c:\Oracle\Middleware\wlserver_12.1\samples\server\medrecd\dist\physicianEar
```

a new directory, `mytestear`, would be created in the staging directory of each server in `mycluster`. If you do not specify a deployment name, a default deployment name (and staging subdirectory) is used:

- For exploded archive deployments, the deployment name and staging subdirectory are the name of the directory you deployed (*`physicianEar`* in the example above).
- For archived deployments, the default deployment name is the name of the archive file without the extension. For example, if you deploy `physicianEar.ear`, the deployment name and staging subdirectory are `physicianEar`.

The WebLogic Remote Console uses stage mode as the default mode when deploying to more than one WebLogic Server instance. `weblogic.Deployer` uses the target server's staging mode as the default, and Managed Servers use stage mode by default.

Stage mode ensures that each server has a local copy of the deployment files on hand, even if a network outage makes the Administration Server unreachable. However, if you are deploying

very large applications to multiple servers or to a cluster, the time required to copy files to target servers can be considerable. Consider nostage mode to avoid the overhead of copying large files to multiple servers.

Syntax for Stage Mode

To use stage mode, specify `-stage` as an option to `weblogic.Deployer`, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
   -password weblogic -name mydeploymentname
   -targets myserver1,myserver2,myserver3 -stage
   -deploy c:\localfiles\myapp.ear
```

Using external_stage Mode Deployment

External_stage mode is similar to stage mode, in that target servers deploy using local copies of the deployment files. However, the Administration Server does not automatically copy the deployment files to targeted servers in external_stage mode; instead, you must copy the files to the staging directory of each target server before deployment. You can perform the copy manually or use automated scripts.

Within each target server's staging directory, deployment files must be stored in a subdirectory that reflects the deployment name. This can either be the name you type in for the deployment, or the default deployment name (the name of the exploded archive directory, or the name of the archive file without its file extension).

External_stage mode is the least common deployment staging mode. It is generally used only in environments that are managed by third-party tools that automate the required copying of files. You may also choose to use external_stage mode when you are deploying very large applications to multiple machines and you do not have a shared file system (and cannot use nostage mode). Using external_stage in this scenario decreases the deployment time because files are not copied during deployment.

You can use the `-noversion` option to turn off the requirement that deployment files be on the Administration Server, but the -noversion option causes versioning information to be ignored so you cannot use the -noversion option with versioned applications. See Common Arguments.

Syntax for external_stage Mode

To deploy an application in external_stage mode:

1. Make sure that the deployment files are accessible to the Administration Server. (See Uploading Deployment Files from a Remote Client.)
2. On each target server for the deployment, create a subdirectory in the staging directory that has the same name as the deployment name. For example, if you will specify the name `myEARExternal` for the deployment name, create a `myEARExternal` subdirectory in the staging directories for each target server.

> **Note:**
>
> If you do not specify a deployment name at deployment time, WebLogic Server selects a default name. See Understanding Default Deployment Names for more information.

3. If you are deploying a versioned application for use with production redeployment, create a subdirectory beneath the application directory for the version of the application you are deploying. For example, if you are deploying `myEARExternal` at version level 2.0Beta, the deployment files must reside in a subdirectory of each target server's staging directory named `myEARExternal\2.0Beta`.
4. Copy the deployment files into the staging subdirectories you created in Step 2 or 3 above.
5. Deploy the application or module using the `weblogic.Deployer` utility. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -name weblogic
   -password weblogic -external_stage -name myEARExternal
   -deploy c:\myapps\myear
```

Changing the Default Staging Behavior for a Server

The server staging mode specifies the default deployment mode for a server if none is specified at deployment time. For example, the server staging mode is used if you deploy an application or standalone module using `weblogic.Deployer` and you do not specify a staging mode. On the Administration Server, the default staging mode is nostage and on Managed Servers, it is stage.

> **Note:**
>
> You can only change the server staging mode by using the WebLogic Remote Console or by directly changing the ServerMBean using JMX.
>
> Changing the server staging mode does not affect existing applications. If you want to change the staging mode for an existing application, you must undeploy the application deployment and then redeploy it with the new staging mode.

To set the server staging mode using the WebLogic Remote Console, in the **Edit Tree**, go to the **Deployments**: **App Deployments**: *myApplication*: **Overview** page. For detailed information on staging modes, see Deploying Applications and Modules with weblogic.Deployer.

Staging Deployment Plans

An application's deployment plan can be staged independently of the application archive, allowing you to stage a deployment plan when the application is not staged. WebLogic Server provides three options for staging deployment plans using `weblogic.Deployer`:

- `-planstage`—Copies the deployment plan to target servers' staging directories.
- `-plannostage`—Does not copy the deployment plan to target servers, but leaves it in a fixed location.
- `-planexternal_stage`—Does not copy the deployment plan to target servers. Instead, you must ensure that the deployment plan has been copied to the correct subdirectory in the target servers' staging directories. You can manually copy the deployment plan or use a third-party tool or script.

If you do not specify a staging mode, the deployment plan uses the value specified for application staging as the default.

For example, if deployment plan staging is not specified and application staging is set to `stage`, the deployment plan staging mode is set to `planstage`.

If both deployment plan staging and application staging are not specified, then the server setting is used as the default application staging mode. For example, if the server setting is to not stage, then the deployment plan staging mode is set to `plannostage`. In this case, if deployment plan staging is required, it must be specified.

Distributing Applications to a Production Environment

Distributing an application prepares it for deployment by copying its deployment files to all target servers and validating it. After you distribute an application, you can start it in administration mode, which restricts access to the application to a configured administration channel and allows you to distribute the application to a production environment (or distribute a new version of an application) without opening the application to external client connections.

While in administration mode, you can connect to an application only via a configured administration channel. This allows you to perform final ("sanity") checking of the application directly in the production environment without disrupting clients. After performing final testing, you can either undeploy the application to make further changes, or start the application to make it generally available to clients.

You can use the `-adminmode` option application in administration mode. See Starting a Distributed Application in Administration Mode.

When deploying across WebLogic domains, the `CrossDomainConnector` role allows you to make inter-domain calls from foreign domains. For a complete listing of all security roles, see Default Global Roles in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.

- Distributing an Application
- Starting a Distributed Application in Administration Mode
- Starting a Distributed Application

Distributing an Application

To distribute an application, use the `weblogic.Deployer` **`-distribute`** command, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
     -password weblogic -distribute -name myTestDeployment
     /myDeployments/myApplication/
```

Starting a Distributed Application in Administration Mode

After WebLogic Server distributes the deployment files, you can start the application in administration mode so that you can access and test it via a configured administration channel. To configure an administration channel, see Configuring Network Resources in *Administering Server Environments for Oracle WebLogic Server*.

To start a distributed application in administration mode, use the `-start` command with the `-adminmode` option:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
     -password weblogic -start -adminmode -name myTestDeployment
     /myDeployments/myApplication/
```

Starting a Distributed Application

After performing final testing of a distributed application using a configured administration channel, you can open the application to new client connections by using the `weblogic.Deployer -start` command without the `-adminmode` option:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
     -password weblogic -start -name myTestDeployment
```

Deploying Shared Jakarta EE Libraries and Dependent Applications

Jakarta EE library support in WebLogic Server provides an easy way to share one or more Jakarta EE modules or JAR files among multiple enterprise applications.

A Jakarta EE library is a standalone Jakarta EE module, multiple Jakarta EE modules packaged in an enterprise application (EAR), or a plain JAR file that is registered with the Jakarta EE application container upon deployment. After a Jakarta EE library has been registered, you can deploy enterprise applications that reference the library. Each referencing application receives a copy of the shared Jakarta EE library module(s) on deployment, and can use those modules as if they were packaged as part of the application itself.

> **Note:**
>
> Oracle documentation and WebLogic Server utilities use the term *library* to refer to both Jakarta EE libraries and optional packages. Optional packages are called out only when necessary.

Understanding Deployment Behavior for Shared Libraries

WebLogic Server supports shared Jakarta EE libraries by merging the shared files with a referencing application when the referencing application is deployed.

You first register a Jakarta EE library or optional package with one or more WebLogic Server instances or clusters by deploying the library and indicating that the deployment is a library (see Registering Libraries with WebLogic Server). Deploying a library or package does not activate the deployment on target servers. Instead, WebLogic Server distributes the deployment files to target servers (if nostage deployment mode is used) and records the location of the deployment files, the deployment name, and any version string information for the library or package.

When an application that references a shared library or package is deployed, WebLogic Server checks the names and version string requirements against the libraries registered with the server. If an exact match for a library or package name is not found, or if the version requirements are not met, the application deployment fails.

If WebLogic Server finds a name and version string match for all of the libraries referenced in the application, the server adds the libraries' classes to the classpath of the referencing application and merges deployment descriptors from both the application and libraries in memory. The resulting deployed application appears as if the referenced libraries were bundled with the application itself.

The contents of a Jakarta EE library are loaded into classloaders in the same manner as any other Jakarta EE modules in an enterprise application. For example, EJB modules are loaded as part of the referencing application's classloader, while Web application modules are loaded in classloaders beneath the application classloader. If a shared Jakarta EE library consists of an EAR, any classes stored in the EAR's `APP-INF/lib` or `APP-INF/classes` subdirectory are also available to the referencing application.

You cannot undeploy any Jakarta EE libraries or optional packages that are referenced by a currently deployed application. If you need to undeploy a shared library or package, you must first undeploy all applications that use the shared files. For regular application maintenance, you should deploy a new version of a shared library or package and redeploy referencing applications to use the newer version of the shared files. See Editing Manifest Entries for Shared Libraries in *Developing Applications for Oracle WebLogic Server* for more information.

Registering Libraries with WebLogic Server

A shared Jakarta EE library is a standard Jakarta EE module or enterprise application that is registered with a WebLogic Server container as a library. To register a Jakarta EE library with a WebLogic Server container, perform the following steps:

1. Ensure that the deployment file(s) you are working with represent a valid Jakarta EE library or optional package. See Creating Shared Jakarta EE Libraries in *Developing Applications for Oracle WebLogic Server*.
2. Select the WebLogic Server targets to which you will register the library or package. Shared libraries must be registered to the same WebLogic Server instances on which you plan to deploy referencing applications. (You may consider deploying libraries to all servers in a domain, so that you can later deploy referencing applications as needed.)
3. Register the library or package by deploying the files to your selected target servers, and identifying the deployment as a library or package with the `-library` option. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
     -password weblogic -deploy -targets myserver1,myserver2
     -library /deployments/myLibraryApplication/
```

Specifying Library Versions at Deployment

As a best practice, your development team should always include version string information for a library or optional package in the manifest file for the deployment. See Editing Manifest Entries for Shared Libraries in *Developing Applications for Oracle WebLogic Server* for more information.

If you are deploying a library or package that does not include version string information, you can specify it at the command line using one or both of the following options:

- `libspecver`—Defines a specification version for the library or package.
- `libimplver`—Specifies an implementation version for the library or package.

For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -username weblogic
     -password weblogic -deploy -targets myserver1,myserver2
```

```
-library -libspecver 700 -libimplversion 7.0.0.1Beta
/deployments/myLibraryApplication/
```

> **Note:**
>
> If both the manifest file and the weblogic.Deployer command line specify version information, and the values do not match, the deployment fails.
>
> If you initially registered a library without providing a specification or implementation version, you must undeploy the library before registering a newer version and specifying version string information.
>
> You can register multiple versions of a library or package, but each version must use the same version strings. For example, you cannot register a library having only a specification version, and then register a new version of the library having both a specification and an implementation version.

Deploying Applications That Reference Libraries

After you deploy the required library and package versions, you can deploy enterprise applications and modules that reference the libraries. Successfully deploying a referencing application requires that two conditions are met:

- All referenced libraries are registered on the application's target servers.
- Registered libraries meet the version requirements of the referencing application.

There is no special syntax for deploying a referencing application. Deploy the application as you would a standard enterprise application or Jakarta EE module.

> **Note:**
>
> Referencing applications can be configured with varying levels of version requirements for shared libraries.

An application might be configured to require on of the following:

- The latest version of a shared library or package (the latest registered version), or
- A minimum version of a library or package (or a higher version), or
- An exact version of a library or package.

If you cannot deploy a referencing application because of version requirements, try registering the required version of the conflicting library or package. Or, consult with your development team to determine whether the version requirements of the application can be relaxed. See Referencing Libraries in an enterprise application in *Developing Applications for Oracle WebLogic Server* for more information.

> **Note:**
>
> If you have an EAR library that contains Web application components, you cannot deploy multiple applications that reference the library because attempting to do so will result in a context path conflict. This is because context-root cannot be set for individual Web applications in EAR libraries; it can only be set for an entire library.

See Overview of Jakarta EE Libraries and Optional Packages in *Developing Applications for Oracle WebLogic Server* for more information.

Best Practices for Deploying Applications

Follow Oracle-recommended best practices when deploying applications.

- The Administration Server in a multiple-server domain should be used only for administration purposes. Although you can deploy to the Administration Server in a multiple-server domain, this practice is not recommended except during development.
- Package deployment files in an archive format (`.ear`, `.jar`, `.war`, and so forth) when distributing files to different users or environments.
- Check the scenarios described under Packaging Files for Deployment before deploying. In many cases it is more convenient to deploy applications in exploded format rather than archive format.
- The WebLogic Remote Console, `weblogic.Deployer` tool, `wldeploy` Ant task, and WLST all provide similar functionality for deploying applications:
  - Use the WebLogic Remote Console for interactive deployment sessions where you do not know the exact location of deployment files, or the exact names of target servers or deployed applications.
  - Use `weblogic.Deployer` to integrate deployment commands with existing administrative shell scripts or automated batch processes.
  - Use `wldeploy` in conjunction with the split development directory for developing and deploying new applications. `wldeploy` can also be used in place of `weblogic.Deployer` in administration environments that use Ant, rather than shell scripts.
  - Use WLST when you want to create automated scripts that perform deployment tasks.
- Use `wldeploy`, rather than the `autodeploy` directory, to deploy your own applications during development. The `autodeploy` directory is best used for quickly running new applications in a test or temporary environment. For example, if you download a sample application and want to evaluate its functionality, the `autodeploy` directory provides a quick way to deploy the application to a development server.

6

Auto-Deploying Applications in Development Domains

Learn about auto-deployment in WebLogic Server, a method for quickly deploying an application to a standalone server (Administration Server) for evaluation or testing. It is recommended that this method be used only in a single-server development environment.

> **Note:**
>
> Oracle recommends that you use the WebLogic Server split development directory and wldeploy ant task, rather than auto-deployment, when developing an application. See Creating a Split Development Directory in *Developing Applications for Oracle WebLogic Server*.

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 Remote Server is started. Auto-deployment deploys only to the Administration Server.

> **Note:**
>
> Due to the file locking limitations of Windows, if an application is exploded, all the modules within the application must also be exploded. In other words, you cannot use auto-deployment with an exploded application or module that contains a JAR file.
>
> Auto-deployment is intended for use with a single server target in a development environment. If you use other tools, such as the WebLogic Remote Console, to add targets to an auto-deployed, exploded application, redeploying the application does not propagate changes to the new target servers.

The following sections provide information on how to use auto-deployment in a development domain:

- Enabling and Disabling Auto-Deployment
  You can run a WebLogic Server domain in two different modes: development and production. Only development mode allows you to use the auto-deployment feature.
- Auto-Deploying, Redeploying, and Undeploying Archived Applications
  Administrators can deploy, redeploy or undeploy archived applications automatically with minimal intervention.

- Auto-Deploying, Redeploying, and Undeploying Exploded Archives
  To auto-deploy an application in exploded archive format, copy the entire exploded archive directory to the `/autodeploy` directory. WebLogic Server automatically deploys exploded archive applications using the nostage deployment mode.
- Limitations of Auto-Deployment
  Auto-deployed applications are very convenient in a development environment, but there are some limitations.

Enabling and Disabling Auto-Deployment

You can run a WebLogic Server domain in two different modes: development and production. Only development mode allows you to use the auto-deployment feature.

Development mode enables a WebLogic Server instance to automatically deploy and update applications that are in the `domain_name/autodeploy` directory (where `domain_name` is the name of a WebLogic Server domain). Production mode disables the auto-deployment feature and prevents any applications you place in the `autodeploy` directory after you switch to production mode from being deployed. When you switch from development mode to production mode, any applications that were previously deployed via the `autodeploy` directory remain deployed; if you wish to undeploy or redeploy such applications after you have switched to production mode, you must undeploy or redeploy them manually (for instance, with the `weblogic.Deployer` command and the `-undeploy` or `-redeploy` options, as described in weblogic.Deployer Command-Line Reference).

By default, a WebLogic Server domain runs in development mode. To specify the mode for a domain, see *Creating WebLogic Domains Using the Configuration Wizard*.

Auto-Deploying, Redeploying, and Undeploying Archived Applications

Administrators can deploy, redeploy or undeploy archived applications automatically with minimal intervention.

To auto-deploy an archived application, copy its archive file to the `/autodeploy` directory. WebLogic Server automatically sets the application's deployment mode to stage mode.

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.

To undeploy an archived deployment unit that was auto-deployed, delete the application from the `/autodeploy` directory. WebLogic Server stops the application and removes it from the configuration.

> **Note:**
>
> If you delete an application from the `/autodeploy` directory when the server is not active, WebLogic Server will not detect that the application was deleted even when the server is again in an active state. In order to prevent an out-of-sync domain tree, Oracle recommends that you only remove applications from the `/autodeploy` directory when the server is in an active state.

Auto-Deploying, Redeploying, and Undeploying Exploded Archives

To auto-deploy an application in exploded archive format, copy the entire exploded archive directory to the `/autodeploy` directory. WebLogic Server automatically deploys exploded archive applications using the nostage deployment mode.

> **Note:**
>
> Due to Windows file locking limitations, if you deploy an exploded EAR directory that contains archived modules (JAR files), the JAR files are locked during the deployment and you will not be able to remove them. Therefore, if an application you plan to auto-deploy is exploded, all of the modules it contains must also be exploded.
>
> Auto-deployment will fail if you attempt to deploy an application in exploded archive format when the contents of the entire exploded archive directory are not in the `/autodeploy` directory. Because large applications in exploded archive format can take a long time to copy into the `/autodeploy` directory, Oracle recommends that you copy the exploded archive directory into the `/autodeploy` directory while the server is in an inactive state. After the entire exploded archive directory has been copied in the `/autodeploy` directory, you can return the server to an active state and the application will be auto-deployed. Alternatively, Oracle recommends that you deploy large applications with weblogic.Deployer, described in weblogic.Deployer Command-Line Reference.

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.

To redeploy files in an exploded application directory:

1. When you first deploy the exploded application directory, create an empty file named `REDEPLOY`, and place it in the `WEB-INF` or `META-INF` directory, depending on the application type you are deploying:

   An exploded enterprise application has a `META-INF` top-level directory which contains the `application.xml` file.

   An exploded Web application has a `WEB-INF` top-level directory which contains the `web.xml` file.

   An exploded EJB application has a `META-INF` top-level directory which contains the `ejb-jar.xml` file.

   An exploded connector has a `META-INF` top-level directory which contains the `ra.xml` file.

> **Note:**
>
> The `REDEPLOY` file works only for an entire deployed application or standalone module. If you have deployed an exploded enterprise application, the `REDEPLOY` file controls redeployment for the entire application—not for individual modules (for example, a Web application) within the enterprise application. If you deploy a Web application by itself as an exploded archive directory, the `REDEPLOY` file controls redeployment for the entire Web application.

2. To redeploy the exploded application, copy the updated files over the existing files in that directory.
3. After copying the new files, modify the `REDEPLOY` file in the exploded directory to alter its timestamp.

When the Administration Server detects the changed timestamp, it redeploys the contents of the exploded directory.

> **Note:**
>
> You must touch the `REDEPLOY` file (alter its timestamp) any time you wish to trigger redeployment of an auto-deployed application. Even if you modify an application while a server is shut down, you must touch `REDEPLOY` to ensure that changes are applied when the server next starts up.

To undeploy an application that was auto-deployed in exploded format, use the `weblogic.Deployer -undeploy` command, or use the WebLogic Remote Console to remove the deployment configuration. Then remove the application files from the `/autodeploy` directory.

> **Note:**
>
> If you delete application files from the `/autodeploy` directory when the server is not active, WebLogic Server will not detect that the application files were deleted even when the server is again in an active state. In order to prevent an out-of-sync domain tree, Oracle recommends that you only remove application files from the `/autodeploy` directory when the server is in an active state.

Limitations of Auto-Deployment

Auto-deployed applications are very convenient in a development environment, but there are some limitations.

- There is no configuration in `config.xml` for an auto-deployed application. Therefore, in the WebLogic Remote Console, there are no notes or target pages associated with an auto-deployed application because there is no backing configuration for that information.

- You cannot associate an auto-deployed application with a deployment plan because auto-deployed applications do not support any configuration operations which would be reflected in a deployment plan.
- You cannot set up security policies or roles for auto-deployed applications.
- You cannot undeploy or redeploy auto-deployed applications using WebLogic Server tools.

7

Using Edition-Based Redefinition (EBR) to Update Applications in a Production Environment

Learn about EBR, a database feature that gives you the flexibility to perform an online update of the database component of an application, thereby ensuring an uninterrupted availability of the application.

This chapter includes the following sections:

- About Edition-Based Redefinition
  With EBR, an application can store two editions of the database schema within the same database, at the same time. One to be used by the pre-upgrade application and the other by the post-upgrade application.
- Enabling Your Application for EBR
  Before you begin EBR-enabling your environment, you must stop the WebLogic Server domain.
- Configuring WebLogic Data Sources to Use Editions
  During an EBR update, WebLogic Server automatically manages the user sessions such that existing user sessions remain connected to the pre-upgrade application (referencing Edition 1 of the database schema) while the new user sessions are connected to the post-upgrade application (referencing Edition 2 of the database schema).
- About Application Software and Domain Configuration Updates
  An application update is typically a two-step process – software (code) update and WebLogic Server domain configuration data update. Depending on your environment, you may use a different update mechanism for each step.
- Synchronizing Application Software and Domain Configuration Updates
  WebLogic Server helps you determine whether a Managed Server should accept the configuration change or not.

About Edition-Based Redefinition

With EBR, an application can store two editions of the database schema within the same database, at the same time. One to be used by the pre-upgrade application and the other by the post-upgrade application.

When an online update is complete, the pre-upgrade application references the earlier edition (Edition 1) of the database schema and the post-upgrade application references the new edition (Edition 2) of the database schema.

This co-existence of multiple editions of the database enables the ongoing user sessions to continue using the pre-upgrade application while the new sessions are directed to the post-upgrade application. After all the existing sessions are complete, the pre-upgrade application is retired and the post-upgrade application becomes fully operational; the entire upgrade process is completed with zero downtime.

For more details about EBR and to familiarize yourself with key terminologies associated with this feature, see Using Edition-Based Redefinition in the *Database Development Guide*.

> **Note:**
>
> You can also perform an application update by using EBR with the WebLogic Zero Downtime Patching (ZDT Patching) feature. However, you have to EBR-enable your application and synchronize the configuration update during the ZDT rollout of updates across a domain. For details, see the following sections:
>
> - Synchronizing Application Software and Domain Configuration Updates
> - Introduction to Zero Downtime Patching in *Administering Zero Downtime Patching Workflows*

Enabling Your Application for EBR

Before you begin EBR-enabling your environment, you must stop the WebLogic Server domain.

See Starting and Stopping Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

You must complete the following key steps to use EBR:

- Enabling users and schema objects to use editions
- Enabling your applications to use editioning views

For more information about readying your application for EBR, see Using EBR to Upgrade an Application in the *Database Development Guide*.

Configuring WebLogic Data Sources to Use Editions

During an EBR update, WebLogic Server automatically manages the user sessions such that existing user sessions remain connected to the pre-upgrade application (referencing Edition 1 of the database schema) while the new user sessions are connected to the post-upgrade application (referencing Edition 2 of the database schema).

To facilitate the automatic management of user sessions, you have to configure the WebLogic data sources to determine the database edition to which it should connect. There are different ways to configure data sources. For more information, see the section "Configuring WebLogic Data Sources to Use Editions", under Using Edition-Based Redefinition in *Administering JDBC Data Sources for Oracle WebLogic Server*.

About Application Software and Domain Configuration Updates

An application update is typically a two-step process – software (code) update and WebLogic Server domain configuration data update. Depending on your environment, you may use a different update mechanism for each step.

A software update, for instance, is done by installing the new product binaries while the configuration data update flows in through the Administration Server. The configuration changes are first communicated to the Administration Server through tools such as WLST,

Management REST API, JMX, and so on. The Administration Server then rolls out the changes to the Managed Servers within the domain.

When performing a rolling update of an application that uses EBR, if the software update requires a concurrent configuration update, the different update mechanisms may lead to a mismatch between the software version and the domain configuration version (old version of the software using the new configuration data or the new version of the software using the old configuration data) resulting in incompatibility issues.

Example 7-1 Illustrates the incompatibility between the software and the domain configuration versions

Let's assume that you have an environment comprising:

- One Administration Server - AS.
- Two Managed Servers - MS1 and MS2.
- The servers are using the software 'S1' and are connected to the database edition 'E1' (pre-upgrade state).
- In the post-upgrade state, the servers are expected to use the new software 'S2' and connect to the new database edition 'E2'.

Now, the Administration Server, the first server in the domain to get updated, receives a configuration change followed by an installation of the updated software S2. The two Managed Servers (MS1 and MS2) continue to be in the preupgrade state (that is, S1-E1 combination). The Administration Server then broadcasts the configuration change to MS1 and MS2. The servers receive the updated configuration data and connect to the new edition (E2) of the database while continuing to use the old software (S1).

At this point, the servers will have the old software configured with the new configuration data (S1-E2 combination). This incompatible state exists until you upgrade the application, typically by performing a rolling restart of the Managed Servers, to have the software updated to S2 in both MS1 and MS2. The servers have now attained a consistent state of using the updated versions of both software and configuration data (S2-E2 combination).

A state of incompatible versions can exist even if you were to consider updating the software first, followed by the configuration data. In this case, the Administration Server is updated with the new software S2, and then you perform a rolling restart of the Managed Servers to have the software updated to S2 in both MS1 and MS2. At this point, the servers use the new software configured with the old configuration data (S2-E1 combination). The Administration Server then receives a configuration update which it broadcasts to the Managed Servers. The servers now attain a consistent state of using the new software configured with the new configuration data (S2-E2 combination).

Irrespective of the sequence of the update (software first or configuration first), there will be an incompatibility issue. When updating applications using EBR, a mismatch between software version and configuration data version is not desirable.

A coordinated and synchronized rolling update can be used to prevent these interim periods of inconsistency between the software and the configuration data.

- Synchronized Rolling Update

Synchronized Rolling Update

During a synchronized rolling update, the configuration data gets updated concurrently as the servers are updated with the new software version, ensuring that there is complete synchronization between the configuration and the software versions on the server.

Example 7-2 Synchronized Rolling Update

Let's assume that you have updated the software and the configuration on the Administration Server (S2-E2). The Managed Servers continue to be in the pre-upgrade state and use the old software (S1) that is communicating with the earlier edition (E1). While they still have the old software, they are not *ready* for the new configuration (that is, the new edition E2).

The Administration Server does not know whether the Managed Servers are ready, and so it assumes that when a configuration change arrives, it should pass on the change to the Managed Servers. However, when the Managed Servers are configured to support synchronization, they ignore the changes until they are ready. For example, if you perform a synchronized rolling restart of MS1 and reboot it with new software, the synchronization logic on MS1 will indicate that MS1 is now ready to accept the previously ignored changes. This means that MS1 has its software updated (from S1 to S2) and uses the new edition (E2) of the database schema to communicate with the updated software (S2).

At this point in time, there are two editions of the database running and the individual servers, MS1 and MS2, are each connected to a different edition - an individual server is configured for a single edition at a time. MS1 is running the new version of the software (S2) and is using the new edition of the database (E2). MS2 is running the old version of the software (S1) and is using the old edition of the database (E1). Although the Managed Servers have mismatched configuration with one another (MS1 – E2 and MS2 – E1), individually they have compatible versions of the software and the database schema (MS1 with S2-E2 and MS2 with S1-E1). This process is called a *synchronized* rolling update - the version of the software is kept synchronized with the version of the WebLogic configuration and, therefore, the database edition.

Likewise, when MS2 gets the new version of the software, it is ready and willingly accepts the new configuration. After the process is complete, all servers have the new version of the software (S2) and point to the new database edition (E2).

To perform a coordinated and synchronized rolling update, WebLogic Server provides built-in mechanisms using which you can ensure that the Managed Server accepts the domain configuration update only after the software update is complete. That is, the Managed Server processes a configuration change only after the updated software is installed on the server. These mechanisms prevent incompatible software and configuration data versions, and ensure that the configuration version and the software version always stay synchronized.

Synchronizing Application Software and Domain Configuration Updates

WebLogic Server helps you determine whether a Managed Server should accept the configuration change or not.

You can use one of the following two checkers depending on how you want to update the Managed Server:

- Lockfile Checker
- Version Checker

These checkers prevent the Managed Servers from accepting configuration changes that are not synchronized with the installed software. Depending on when these checkers get initiated, they are categorized into:

- **Configuration Independent** (Example: Lockfile Checker): Any change in the environment will initiate the checker (not necessarily a configuration change). Depending on the nature of the change, the checker instructs the Managed Server to ignore or accept the change.

- **Configuration Based** (Example: Version Checker): Any configuration update will initiate the checker. Depending on what the change is, the checker instructs the Managed Server to accept or reject the change.

The checkers get initiated when the Managed Server receives an indication from the Administration Server of a possible change, during the following events:

- When the Administration Server accepts a configuration change and passes it over to the Managed Server.
- When you restart the Managed Server and it detects updates from the Administration Server.
- When the Managed Server receives periodic signals from the Administration Server indicating a change.

In addition to these checkers, WebLogic Server also enables you to write your own Java code plug-ins. See Writing and Configuring Your Own Java Plug-In.

Lockfile Checker

The configuration independent Lockfile Checker gets initiated when the Managed Server expects a configuration change coming from the Administration Server. Set the Lockfile on the machine where the Managed Server is running. For as long as the Lockfile exists, the Managed Server does not accept the configuration change from the Administration Server. After the software is updated on the Managed Server, remove the Lockfile. As soon as the Lockfile is removed, the configuration change is accepted and applied by the Managed Server.

Version Checker

The configuration based Version Checker, on the other hand, gets initiated when the Managed Server receives a configuration update. This checker enables the Managed Server to decide the version of the software for which it should accept the configuration data change. To achieve this, you need to include a file that contains the version of the software, as part of the software installation process.

When the Managed Server receives the configuration update from the Administration Server, it compares the version (a numerical value) that is part of the Version Checker with the software version included in the installation file and accepts the change only if the two versions match.

Configuring the Lockfile Checker

The Lockfile Checker gets called when a Managed Server expects a configuration change from the Administration Server. To make use of this checker, you have to configure it at the ‘Expected Change’ hook point.

> **Note:**
>
> The assumption is that your environment consists of an Administration Server and a Managed Server running on different machines. Therefore, the servers have their own copies of the `config.xml` file.

1. Use WLST to connect to the Administration Server:

   ```
   connect('<username>', 'ok', 't3://localhost:7001')
   ```

2. Create a callout to refer to the Lockfile Checker, as shown below:

   ```
   edit()
   startEdit()
   cd('/')
   callout1=cmo.createCallout('LockFileChecker-1')
   callout1.setHookPoint("HOOK_POINT_EXPECTED_CHANGE")
   callout1.setClassName("weblogic.configuration.ConfigurationLockfileChecker"
   )
   callout1.setArgument("<Location of the Lock File in the Managed Server>")
   activate()
   ```

Example 7-3 Illustrates the use of the Lockfile Checker

1. Use WLST to connect to the Administration Server, as described in step 1 above.
2. Create the callout on the Administration Server node and activate it to push the callout changes to the Managed Server.
3. Create a file named Lockfile on the machine where the Managed Server is running (in the location that you have specified as the argument of the callout when configuring the Lockfile Checker). Now, the Managed Server will not accept any configuration changes from the Administration Server.
4. Using the WebLogic Remote Console, in the **Edit Tree**, go to **Environment**, then **Servers**.
5. Select the Managed Server whose configuration you want to change.
6. Go to the **Advanced**: **Concurrency** page and change the value of **Max Concurrent New Threads** from `50` to `100`.
7. Save and commit the changes.
8. Check the `config.xml` file for the Managed Server or connect to the Managed Server by using WLST, and then observe the value of `Max Concurrent New Threads`.
   The value is not updated on the Managed Server because of the presence of the Lockfile.
9. Now, remove the Lockfile from the Managed Server and check the `config.xml` file for the Managed Server or connect to the server by using WLST.
   Observe the value of `Max Concurrent New Threads`. The value is now updated to `100`.

Configuring the Version Checker

The Version Checker gets called when a Managed Server receives a configuration update from the Administration Server. To make use of this checker, you have to configure it at the ‘Change Received’ hook point.

> **Note:**
>
> The assumption is that your environment consists of an Administration Server and a Managed Server running on different machines. Therefore, the servers have their own copies of the `config.xml` file.

1. Use WLST to connect to the Administration Server:

   ```
   connect('<username>', '<password>', 't3://localhost:7001')
   ```

2. Change the software version installed on the domain to say, 1.1:

   ```
   edit()
   startEdit()
   cd('/')
   cmo.setInstalledSoftwareVersion('1.1')
   activate()
   ```

3. Create a callout to refer to the Version Checker, as shown below:

   ```
   edit()
   startEdit()
   cd('/')
   callout1=cmo.createCallout('VersionChecker-1')
   callout1.setHookPoint("HOOK_POINT_CHANGE_RECEIVED")
   callout1.setClassName("weblogic.configuration.ConfigurationVersionChecker")
   callout1.setArgument("<Location of the Version File on the Managed
   Server>")
   activate()
   ```

Example 7-4 Illustrates the use of the Version Checker

1. Use WLST to connect to the Administration Server, as described in step 1 above.
2. Create the callout on the Administration Server node and activate it to push the callout changes to the Managed Server.
3. Create the Version Checker file (in the location that you have specified as the argument of the callout at the time of configuring the Version Checker, for example `/MyInstallation/Servers/MS1/VersionChecker`) on the machine where the Managed Server is running. The file should contain the version number.

   ```
   echo "1.0"  >  </MyInstallation/Servers/MS1/VersionChecker >
   ```

4. Using the WebLogic Remote Console, in the **Edit Tree**, go to **Environment**, then **Servers**.
5. Select the Managed Server whose configuration you want to change.
6. Go to the **Web Services**: **Buffering** page and change the value of **Retry Count** to `999`.
7. Save and commit the changes.
8. Check the `config.xml` file for the Managed Server or connect to the Managed Server by using WLST and observe the value of `retry-count`.

Notice that the value is not updated to `999`. This occurs because the version of the installed software (1.1) as specified in the Version Checker file does not match with the expected version set in the callout (1.0).

9. Modify the version number in the Version Checker file to contain the same version as the version configured in the callout. This is the point where the installed software is to be updated before updating the version in the Version Checker file.

```
echo "1.1" >  <Location of the Version Checker>
```

10. Check the `config.xml` file for the Managed Server or connect to the server by using WLST.
Observe that the value of `retry-count` is now updated to `999` because the two versions matched.

Writing and Configuring Your Own Java Plug-In

In addition to the Lockfile Checker and the Version Checker, WebLogic Server also gives you the flexibility to write your own custom logic for deciding whether to accept or reject configurations. In fact, both Lockfile and Version checkers are simply Java plug-ins themselves.

There are two types of Java plug-ins: *Expected Change* and *Change Received*. The difference is that while the *Change Received* plug-in gives the ability to look at the configuration changes to make the accept/reject decision, the *Expected Change* plug-in does not. However, the former requires less computation on both WebLogic's part and the plug-in's part.

Implementing the plug-in requires overriding the *init* and *callout* methods from `weblogic.callout.spi.WebLogicCallout.java` .

To install the plug-in, do the following:

1. Add the Java plug-in classes to the Managed Server classpaths and restart the servers.
2. Create the callouts using WLST, as described for the out-of-the-box checkers. See Configuring the Lockfile Checker and Configuring the Version Checker.

ConfigurationLockfileChecker is an *Expected Change* plug-in. The code below does not have access to the `WebLogic DomainMBean`. It simply makes its decision based on the state of the machine on which it is running (that is, the Managed Server on which it is installed) and checks for the presence of the configured file.

```
package weblogic.configuration;

import java.io.File;
import java.util.Map;
import weblogic.callout.spi.WebLogicCallout;
/*
 This is provided out of box for the usage of orchestrating configuration
updates and rollout.
 To use this, user will need to create a Callout under domain,  with the
following parameter

     name:        callout-0
     hookpoint:   HOOK_POINT_EXPECTED_CHANGE
     classname:   weblogic.configuration.ConfigurationLockfileChecker
     argument:    /wls-installation/servers/Server-0/CONFIGURATION_LOCKFILE
```

```
     both hookpoint and classname should be as above,  argument is the path
to the configuration lockfile.  You can name
     it any way you want.
     This class will be invoked whenever the following occurs:
      #1. managed server boots up,
      #2. managed server does periodic 'heartbeat' for getting any changes
from the admin server
      #3. admin server commit a configuration transaction and send the bits
to this managed server

      The existence of this lockfile is checked.  If the lockfile exists, the
managed server configuration will be updated,
      otherwise, the change will be ignored until the next time, one of the
above 3 situation occurs.

 * Copyright (c) 2019, Oracle and/or its affiliates. All rights reserved.
 */
public class ConfigurationLockfileChecker implements WebLogicCallout {

  File lockfile = null;

  @Override
  public String callout(String hookPoint, String location, Map<String,
Object> values) {


    if (WebLogicCallout.HOOK_POINT_EXPECTED_CHANGE.equals(hookPoint)) {
      if (lockfile != null && lockfile.exists()) {
        return WebLogicCallout.RESPONSE_IGNORE;
      }
      return WebLogicCallout.RESPONSE_ACCEPT;
    }

    return WebLogicCallout.RESPONSE_CONTINUE;
  }

  public void init(String argument) {
    if (argument != null) {
      lockfile = new File(argument);
    }
  }
}
```

ConfigurationVersionChecker is a *Change Received* plug-in. The code below looks at both the local machine and the `WebLogic DomainMBean` to make its decision. It checks if the configured version (in the `WebLogic DomainMBean`) matches the version of the software installed on the Managed Server.

```
package weblogic.configuration;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
```

```
import java.util.Map;
import weblogic.callout.spi.WebLogicCallout;
import weblogic.management.configuration.DomainMBean;

/**
 * Copyright (c) 2019, Oracle and/or its affiliates. All rights reserved.
 */

/*
     This is provided out of box for the usage of orchestrating configuration
updates and rollout.
     To use this, user will need to create a Callout under the domain,  with
the following parameter
       name:        callout-0
       hookpoint:   HOOK_POINT_CHANGE_RECEIVED
       classname:   weblogic.configuration.ConfigurationVersionChecker
       argument:    /wls-installation/servers/Server-0/
CONFIGURATION_VERSION_CHECKER

     both hookpoint and classname should be as above,  argument is the path
to the configuration version checker file.
     You can name it any way you want.

     You will also need to set the domain's <installed-software-version> for
using this callout.

     This class will be invoked whenever the following occurs:
      #1. managed server boots up,
      #2. managed server does periodic 'heartbeat' for getting any changes
from the admin server
      #3. admin server commit a configuration transaction and send the bits
to this managed server

      If this configuration version checker exists, the content of this
file,  which should be a version nmuber  eg, "1.0"
      will be used to compare with the domain's <installed-software-version>
property, eg "1.1".  If it matches, managed server's
      configuration will be updated,  otherwise, the change will be ignored
until the next time, one of the above 3 situation
      occurs.
 */

public class ConfigurationVersionChecker implements WebLogicCallout {

  File versionFile = null;

  @Override
  public String callout(String hookPoint, String location, Map<String,
Object> values) {
    if (WebLogicCallout.HOOK_POINT_CHANGE_RECEIVED.equals(hookPoint)) {
      String versionFromFile = getVersionFromFile();
      String pendingVersion =
getVersionFromDomainMBean(values.get(WebLogicCallout.VALUES_PENDING));
      if (versionFromFile != null && pendingVersion != null){
        return versionFromFile.equals(pendingVersion) ?
WebLogicCallout.RESPONSE_ACCEPT: WebLogicCallout.RESPONSE_IGNORE;
```

```
      }
    }
    return WebLogicCallout.RESPONSE_CONTINUE;
  }

  @Override
  public void init(String argument) {
    if (argument != null){
      versionFile = new File(argument);
    }
  }

  String getVersionFromFile() {
    if (versionFile != null && versionFile.exists()) {
      try (FileReader fileReader = new FileReader(versionFile)) {
        BufferedReader bufferedReader = new BufferedReader(fileReader);
        StringBuffer stringBuffer = new StringBuffer();
        String readLine;
        while( (readLine = bufferedReader.readLine()) != null) {
          stringBuffer.append(readLine);
        }
        return stringBuffer.toString();
      } catch (FileNotFoundException e) {
      } catch (IOException e) {
      }
    }
    return null;
  }

  String getVersionFromDomainMBean(Object domainMBean) {
    if (domainMBean != null && domainMBean instanceof DomainMBean) {
      return ((DomainMBean)domainMBean).getInstalledSoftwareVersion();
    }
    return null;
  }
}
```

8

Redeploying Applications in a Production Environment

Learn how to use WebLogic Server redeployment to safely update applications and parts of applications that you have deployed in a production environment.
This chapter includes the following sections:

- Overview of Redeployment Strategies
  In a production environment, deployed applications frequently require 24x7 availability in order to provide uninterrupted services to customers and internal clients. WebLogic Server provides flexible redeployment strategies to help you update or repair production applications based on their required level of availability.
- Understanding When to Use Different Redeployment Strategies
  Review each WebLogic Server redeployment strategy and the scenarios in which you would use each strategy.
- Using Production Redeployment to Update Applications
  WebLogic Server enables you to redeploy a new, updated version of a production application without affecting existing clients of the application, and without interrupting the availability of the application to new client requests.
- Distributing a New Version of a Production Application
  When you distribute a new version of an application, WebLogic Server prepares the new application version for deployment. You can then deploy the application in administration mode, which makes it available only via a configured administration channel. External clients cannot access an application that has been distributed and deployed in administration mode.
- Using In-Place Redeployment for Applications and Standalone Modules
  In-place redeployment immediately undeploys and replaces a running application's deployment files with updated deployment files.
- Using Partial Redeployment for Jakarta EE Module Updates
  The `weblogic.Deployer` utility uses a different command form if you want to redeploy individual modules of a deployed enterprise application. To redeploy a subset of the modules of an enterprise application, specify *`modulename@servername`* in the target server list to identify the modules you want to redeploy.
- Updating Static Files in a Deployed Application
  In a production environment, you may occasionally need to refresh the static content of a Web application module—HTML files, image files, JSPs, and so forth—without redeploying the entire application. If you deployed a Web application or an enterprise application as an exploded archive directory, you can use the `weblogic.Deployer` utility to update one or more changed static files in-place.
- Updating the Deployment Configuration for an Application
  After you have deployed an application or standalone module, you can change the WebLogic Server deployment configuration using either the WebLogic Remote Console or `weblogic.Deployer`.

Overview of Redeployment Strategies

In a production environment, deployed applications frequently require 24x7 availability in order to provide uninterrupted services to customers and internal clients. WebLogic Server provides flexible redeployment strategies to help you update or repair production applications based on their required level of availability.

The following sections describe these strategies in detail.

- Production Redeployment
- In-Place Redeployment

Production Redeployment

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 enables you to update and redeploy an application in a production environment without stopping the application or otherwise interrupting the application's availability to clients. Production redeployment saves you the trouble of scheduling application downtime, setting up redundant servers to host new application versions, manually managing client access to multiple application versions, and manually retiring older versions of an application (see Redeploying Applications and Modules In-Place for information about these manual steps).

Production redeployment can be used in combination with the `-distribute` command to prepare a new version of an application for deployment. Then, you can deploy the application in administration mode, which allows you to perform final sanity testing of a new application version directly in the production environment before making it available to clients. See Distributing a New Version of a Production Application.

Production redeployment is supported only for certain Jakarta EE application types, see Restrictions on Production Redeployment Updates. For production redeployment procedures and related information, see Using Production Redeployment to Update Applications.

In-Place Redeployment

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 standalone Jakarta EE 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.

> **Note:**
>
> In-place redeployment is the redeployment strategy used in previous versions of WebLogic Server.

WebLogic Server uses the in-place redeployment strategy for Jakarta EE applications that do not specify a version identifier, and for Jakarta EE applications and standalone modules that are not supported in Production Redeployment. You should ensure that in-place redeployment of applications and standalone Jakarta EE modules takes place only during scheduled downtime for an application, or when it is not critical to preserve existing client connections to an application. See Using In-Place Redeployment for Applications and Standalone Modules for more information.

WebLogic Server uses the in-place redeployment strategy when performing partial redeployment of a deployed application or module. Partial redeployment can replace either static files in an application, or entire Jakarta EE modules within an enterprise application deployment, as described in Partial Redeployment of Static Files.

- Partial Redeployment of Static Files
- Partial Redeployment of Jakarta EE Modules

Partial Redeployment of Static Files

WebLogic Server enables you to redeploy selected files in a running application, rather than the entire application at once. This feature is generally used to update static files in a running Web application, such as graphics, static HTML pages, and JSPs. Partial redeployment is available only for applications that are deployed using an exploded archive directory.

Partial redeployment of static files does not affect existing clients of the application. WebLogic Server simply replaces the static files for the deployed application, and the updated files are served to clients when requested. See Updating Static Files in a Deployed Application.

Partial Redeployment of Jakarta EE Modules

Partial redeployment also enables you to redeploy a single module or subset of modules in a deployed enterprise application. Again, partial deployment is supported only for applications that are deployed using an exploded archive directory.

Note that redeployment for modules in an enterprise application uses the in-place redeployment strategy, which does not guarantee uninterrupted client access to the module. For this reason, you should ensure that partial redeployment of Jakarta EE modules in an EAR takes place only during scheduled application downtime, or when it is not critical to preserve client access to the application. See Using Partial Redeployment for Jakarta EE Module Updates.

Understanding When to Use Different Redeployment Strategies

Review each WebLogic Server redeployment strategy and the scenarios in which you would use each strategy.

Table 8-1 Summary of Redeployment Strategies

Redeployment Strategy	Summary	Usage
Production Redeployment	Redeploys a newer version of an application alongside an existing version of the application.	• Upgrading Web applications and enterprise applications that demand uninterrupted client access.

Table 8-1 (Cont.) Summary of Redeployment Strategies

Redeployment Strategy	Summary	Usage
In-Place Redeployment of Applications and Modules	Application classloaders are immediately replaced with newer classloaders to load the updated application class files. WebLogic Server does not guarantee uninterrupted client access during redeployment, and existing clients' state information may be lost.	• Replacing applications that have been taken off-line for scheduled maintenance. • Upgrading applications that do not require uninterrupted client access.
Partial Redeployment of Static Files (In-Place Redeployment)	HTML, JSPs, graphics files, or other static files are immediately replaced with updated files.	• Updating individual Web application files that do not affect application clients.
Partial Redeployment of Jakarta EE Modules (In-Place Redeployment)	Module classloaders are immediately replaced with newer classloaders to load the updated class files. WebLogic Server does not guarantee uninterrupted client access to the module during redeployment, and existing clients' state information may be lost.	• Replacing a component of an enterprise application that has been taken off-line for scheduled maintenance, or that does not require uninterrupted client access.

Using Production Redeployment to Update Applications

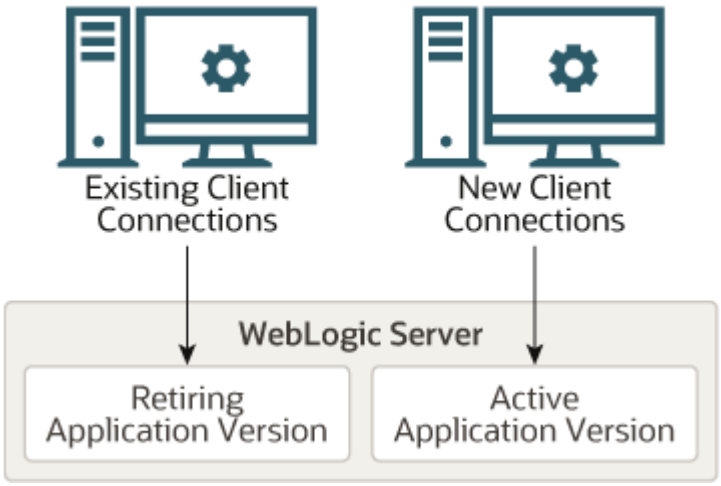
WebLogic Server enables you to redeploy a new, updated version of a production application without affecting existing clients of the application, and without interrupting the availability of the application to new client requests.

The following sections describe using production redeployment to update applications, in detail.

How Production Redeployment Works

WebLogic Server performs production redeployment by deploying a new version of an application alongside an older, running version of the application. While redeployment is taking place, one version of the application "active," while the other version is "retiring." The active application version receives all new client connection requests for the application, while the retiring application version processes only those client connections that existed when redeployment took place. WebLogic Server undeploys the retiring application version after all existing clients of the application have finished their work, or when a configured timeout is reached.

Figure 8-1 Production Redeployment



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.

The retired application is deactivated after the completion of active sessions and in-progress transactions. The deactivated application continues to exist in the server until it is removed using an undeployment method specified in Managing Deployed Applications.

You can roll back the production redeployment process by making the older application version active. This may be necessary if, for example, you determine that there is a problem with the newer version of the application, and you want WebLogic Server to begin moving clients back to the older version. To make the older application version active, redeploy it.

Production Redeployment In Clusters

In a WebLogic Server cluster, each clustered server instance retires its local deployment of the retiring application version when the current workload is completed. This means that an application version may be retired on some clustered server instances before it is retired on other servers in the cluster. Note, however, that in a cluster failover scenario, client failover requests are always handled by the same application version on the secondary server, if the application version is still available. If the same application version is not available on the secondary server, the failover does not succeed.

Requirements and Restrictions for Production Redeployment

In order to use the production redeployment feature, an application must meet certain requirements during the development and deployment phases.

- Development Requirements
- Deployment Requirements
- Restrictions on Production Redeployment Updates

Development Requirements

The production redeployment strategy is supported for:

- Standalone Web Application (WAR) modules and enterprise applications (EARs) whose clients access the application via a Web application (HTTP).
- Enterprise applications that are accessed by inbound JMS messages from a global JMS destination, or from inbound JCA requests.
- All types of Web Services, including conversational and reliable Web Services, but not 8.x Web Services.
- Coherence cache clients – Grid archive (GAR) modules that are deployed within an EAR to storage-disabled managed Coherence servers. Classes in the GAR must be backward compatible with the existing deployment.

Production redeployment is *not* supported for:

- Standalone EJB or RAR modules. If you attempt to use production redeployment with such modules, WebLogic Server rejects the redeployment request. To redeploy such modules, remove their version identifiers and explicitly redeploy the modules.
- Applications that use JTS drivers. For more information on JDBC application module limitations, see JDBC Application Module Limitations in *Administering JDBC Data Sources for Oracle WebLogic Server*.
- Applications that obtain JDBC data sources via the DriverManager API; in order to use production redeployment, an application must instead use JNDI to look up data sources.
- Applications that include EJB 1.1 container-managed persistence (CMP) EJBs. To use production redeployment with applications that include CMP EJBs, use EJB 2.x CMP instead of EJB 1.1 CMP.
- Coherence cache servers – GAR modules that are deployed to storage-enabled managed Coherence servers.

Production redeployment only supports HTTP clients and RMI clients (see Graceful Shut Down of RMI Client Request Processing). Your development and design team must ensure that applications using production redeployment are not accessed by an unsupported client. WebLogic Server does not detect when unsupported clients access the application, and does not preserve unsupported client connections during production redeployment.

During development, applications must be designed to meet specific requirements in order to ensure that multiple versions of the application can safely coexist in a WebLogic Server domain during production redeployment. See Developing Versioned Applications for Production Redeployment in *Developing Applications for Oracle WebLogic Server* for information about the programming conventions required for applications to use production redeployment.

If an enterprise application includes a JCA resource adapter module, the module:

- Must be JCA 1.5 compliant
- Must implement the `weblogic.connector.extensions.Suspendable` interface
- Must be used in an application-scoped manner, having `enable-access-outside-app` set to `false` (the default value).

Before resource adapters in a newer version of the EAR are deployed, resource adapters in the older application version receive a callback. WebLogic Server then deploys the newer application version and retires the entire older version of the EAR.

For a complete list of production redeployment requirements for resource adapters, see Production Redeployment in *Developing Resource Adapters for Oracle WebLogic Server*.

> **Note:**
>
> Because the production redeployment strategy requires an application to observe certain programming conventions, use production redeployment only with applications that are approved by your development and design staff. Using production redeployment with an application that does not follow Oracle's programming conventions can lead to corruption of global resources or other undesirable application behavior.

Deployment Requirements

A deployed application must specify a version number before you can perform subsequent production redeployment operations on the application. In other words, you cannot deploy a non-versioned application, and later perform production redeployment with a new version of the application.

Restrictions on Production Redeployment Updates

WebLogic Server can host a maximum of two different versions of an application at one time.

When you redeploy a new version of an application, you cannot change:

- An application's deployment targets
- An application's security model
- A Web application's persistent store settings

To change any of the above features, you must first undeploy the active version of the application.

Specifying an Application Version Identifier

WebLogic Server attempts to use the production redeployment strategy when the currently-deployed application and the redeployed application specify different versions.

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. Your development process should automatically increment the version identifier with each new application release before packaging the application for deployment.

Maintaining version information in this manner ensures that the production redeployment strategy is used with each redeployment of the application in a production domain. See Developing Versioned Applications for Production Redeployment in *Developing Applications for Oracle WebLogic Server*.

- Assigning a Version Identifier During Deployment and Redeployment

Assigning a Version Identifier During Deployment and Redeployment

If you are testing the production redeployment feature, or you want to use production redeployment with an application that does not include a version string in the manifest file,

specify a unique version string by using the `-appversion` option when deploying or redeploying an application:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -deploy -name myTestDeployment
     -source /myDeployments/myApplication/91Beta
     -targets myCluster -stage -appversion .91Beta
```

The version string specified with `-appversion` is applied only when the deployment source files do not specify a version string in `MANIFEST.MF`. See Application Version Conventions in *Developing Applications for Oracle WebLogic Server* for information about valid characters and character formats for application version strings.

> **Note:**
>
> Do not use `-appversion` to deploy or redeploy in a production environment unless you are certain the application follows Oracle's programming conventions for production redeployment. See Requirements and Restrictions for Production Redeployment. Using production redeployment with an application that does not follow the programming conventions can cause corruption of global resources or other undesirable application behavior.

Displaying Version Information for Deployed Applications

The WebLogic Remote Console displays both version string information and state information for all deployed applications and modules. To view this information, select the **Application Runtime Data** Dashboard in the **Monitoring Tree**.

You can also display version information for deployed applications from the command line using the `weblogic.Deployer` **`-listapps`** command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -listapps
```

Redeploying a New Version of an Application

To redeploy a new version of an application using the production redeployment strategy:

1. Verify that only one version of the application is currently deployed in the domain. See Displaying Version Information for Deployed Applications.
2. Verify that the deployed application and the newer application have different version strings:
   a. Use the instructions in Displaying Version Information for Deployed Applications to determine the currently-deployed version of the application.
   b. Examine the version string in the `MANIFEST.MF` file of the new application source you want to deploy:

      ```
      jar xvf myApp.ear MANIFEST.MF
      cat MANIFEST.MF
      ```
3. Place the new application deployment files in a suitable location for deployment. Oracle recommends that you store each version of an application's deployment files in a unique subdirectory.

For example, if the currently-deployed application's files are stored in:

```
/myDeployments/myApplication/91Beta
```

You would store the updated application files in a new subdirectory, such as:

```
/myDeployments/myApplication/1.0GA
```

> **Note:**
>
> For nostage or external_stage mode deployments, do not overwrite or delete the deployment files for the older version of the application. The original deployment files are required if you later choose to roll back the retirement process and revert to the original application version.

4. Redeploy the new application version and specify the updated deployment files. If the updated deployment files contain a unique version identifier in the manifest file, use a command similar to:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -redeploy -name myTestDeployment
     -source /myDeployments/myApplication/1.0GA
```

> **Note:**
>
> Do not specify the `-source` option for existing versioned applications if the source location has *not* changed. In that case, you must first undeploy, then deploy the application.
>
> See Redeploy.

If the new deployment files do not contain a version identifier in the manifest, see Assigning a Version Identifier During Deployment and Redeployment.

By default WebLogic Server makes the newly-redeployed version of the application active for processing new client requests. Existing clients continue to use the older application until their work is complete and the older application can be safely undeployed.

If you want to specify a fixed time period after which the older version of the application is retired (regardless of whether clients finish their work), use the `-retiretimeout` option with the **`-redeploy`** command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -redeploy -name myTestDeployment
     -source /myDeployments/myApplication/1.0GA
     -retiretimeout 300
```

`-retiretimeout` specifies the number of seconds after which the older version of the application is retired. You can also retire the older application immediately by using the **`-undeploy`** command and specifying the older application version, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -undeploy -name myTestDeployment
     -appversion .91Beta
```

5. Verify that both the old and new versions of the application are deployed, and that the correct version of the application is active. See Displaying Version Information for Deployed Applications.

Undeploying a Retiring Application

If WebLogic Server has not yet retired an application version, you can immediately undeploy the application version without waiting for retirement to complete. This may be necessary if, for example, an application remains in the retiring state with only one or two long-running client sessions that you do not want to preserve. To force the undeployment of a retiring version of an application, use the **`-undeploy`** command and specify the application version:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -undeploy -name myTestDeployment
     -appversion .91Beta
```

> **Note:**
>
> You cannot specify the `-graceful` option to the **`-undeploy`** command when undeploying an application version that is being retired, or waiting for a retirement timeout to occur.
>
> If you do not explicitly specify an application version with the `-appversion` option, WebLogic Server undeploys the active version and all retired versions of the application. If an older version of the application is not yet retired and you run the `-undeploy` command without specifying the `-appversion` option, WebLogic Server logs a warning message in the server log and does not undeploy the unretired version. To later undeploy such versions of the application, you must run the `-undeploy` command again and specify the application version with the `-appversion` option.

Rolling Back the Production Redeployment Process

Reversing the production redeployment process switches the state of the active and retiring applications and redirects new client connection requests accordingly. Reverting the production redeployment process might be necessary if you detect a problem with a newly-deployed version of an application, and you want to stop clients from accessing it.

To roll back the production redeployment process, issue a second **`-redeploy`** command and specify the deployment source files for the older version, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -redeploy -name myTestDeployment
     -source /myDeployments/myApplication/91Beta
     -retiretimeout 300
```

> **Note:**
>
> When you use the `-redeploy` command with `-appversion` option, ensure to include the `-source` option along with it. You cannot redeploy an application with only the `-appversion` option.

If the deployment files do not contain a version identifier in the manifest, see Assigning a Version Identifier During Deployment and Redeployment.

Graceful Shut Down of RMI Client Request Processing

You can use the `-rmigraceperiod` attribute to specify a grace period (in seconds) for processing of RMI client requests when retiring (`-retirement`) or gracefully (`-graceful`) shutting down an application. The work manager of a server instance accepts and schedules RMI calls until the grace period expires without a receiving new RMI client request. If a new RMI client request occurs within the grace period, the grace period is reset and RMI client processing continues until:

- Another RMI request resets the grace period or
- The grace period expires without an RMI client request.

If an RMI client tries to access a retired application or a shutdown application, the client receives `NoSuchObjectException` when trying to invoke the object. If a new version of the object is available, an application needs to catch the `NoSuchObjectException` exception and look up the new version of the object using the JNDI environment property `weblogic.jndi.WLContext.RELAX_VERSION_LOOKUP` to return bindings from the active version of the application.

For example, the following command places the application in administration mode after allowing any pending HTTP sessions or in-process work to complete:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -stop -adminmode -graceful
   -rmigraceperiod 30
```

If graceful retirement is initiated at T0 seconds, and a RMI request (msg1) arrives at T1 seconds where T1 < (T0 + 30), the application waits for T1 + 30 seconds for additional RMI client requests.

- If msg2 arrives at T2 seconds, where T2 < (T1 +30), msg2 is scheduled and the grace period is reset to T2+30 seconds.
- If there are no additional requests or a request (msg2) arrives at T2 seconds, where T2 > (T1 +30), the work manager stops scheduling RMI requests.

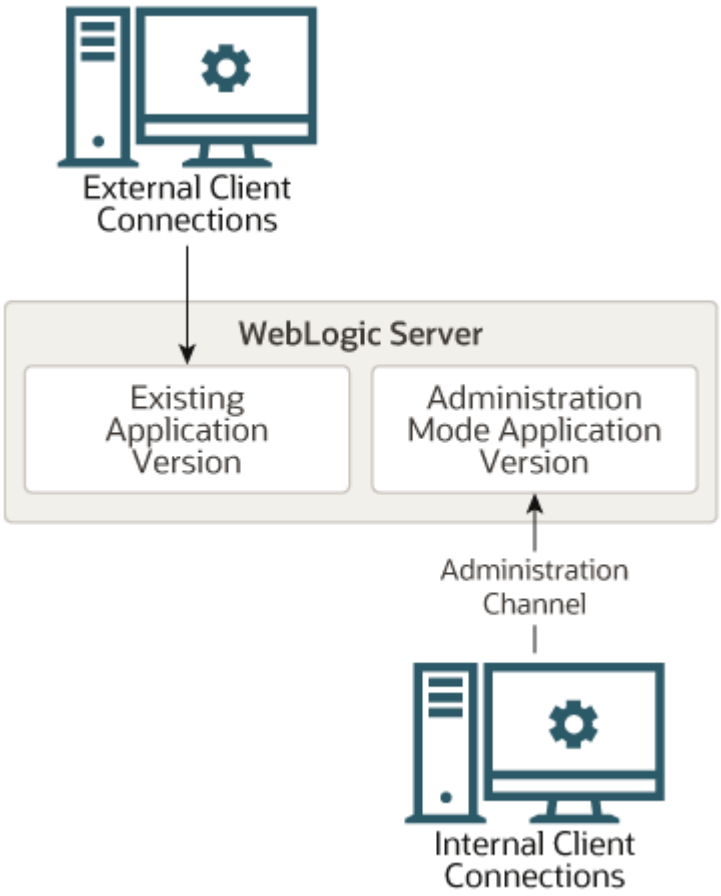
Distributing a New Version of a Production Application

When you distribute a new version of an application, WebLogic Server prepares the new application version for deployment. You can then deploy the application in administration mode, which makes it available only via a configured administration channel. External clients cannot access an application that has been distributed and deployed in administration mode.

You can use the `-adminmode` option to start the application in administration mode. See Starting a Distributed Application in Administration Mode.

The older version of the application remains active to process both new and existing client requests. WebLogic Server does not automatically retire the older version of the application when you distribute and deploy a newer version in administration mode.

Figure 8-2 Distributing a New Version of an Application



After you complete testing of the new application via an administration channel, you either undeploy the new application version or start it. Starting the application causes WebLogic Server to route new client connections to the updated application, and begin retiring the older application version.

- Steps for Distributing a New Version of an Application
- Making an Application Available to Clients
- Best Practices for Using Production Redeployment

Steps for Distributing a New Version of an Application

The basic steps for distributing a new version of an application in administration mode are nearly the same as those documented in Redeploying a New Version of an Application. You simply use the `weblogic.Deployer` **`-distribute`** command, rather than the **`-redeploy`** command, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -distribute -name myTestDeployment
     -source /myDeployments/myApplication/1.0GA
     -appversion 1.0GA
```

Once the application is distributed, start it in administration mode:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -start -adminmode -name myTestDeployment
     -source /myDeployments/myApplication/1.0GA
     -appversion 1.0GA
```

Starting the application in administration mode makes it available only via a configured administration channel. See Configuring Network Resources in *Administering Server Environments for Oracle WebLogic Server*.

Optionally, specify the retirement policy or timeout period for distributed applications.

Making an Application Available to Clients

After performing final testing using the configured administration channel, you can open the distributed version of the application that is running in administration mode to new client connections by using the `weblogic.Deployer` **`-start`** command without the `-adminmode` option:

1. Use the WebLogic Remote Console to view the version and state information of both application versions:
   a. Verify that both versions of the application are still deployed.
   b. Note the version identifier of the application version that is running in administration mode.
2. Use the `-appversion` option to `weblogic.Deployer` to start the application that was distributed and deployed in administration mode:

   ```
   java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
        -password weblogic -start -name myTestDeployment
        -appversion .91Beta
   ```

   By default WebLogic Server routes new client requests to the application version that was previously distributed and running in administration mode. Existing clients continue using the older application until their work is complete and the older application can be safely undeployed. If you want to specify a fixed time period after which the older version of the application is retired (regardless of whether clients finish their work), use the `-retiretimeout` option:

   ```
   java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
        -password weblogic -start -name myTestDeployment
        -appversion .91Beta -retiretimeout 300
   ```

   `-retiretimeout` specifies the number of seconds after which the older version of the application is retired.
3. Use the WebLogic Remote Console to verify that the previously-distributed application is now active, and that the former application version is now retiring. See Displaying Version Information for Deployed Applications.

Best Practices for Using Production Redeployment

When using production redeployment, keep in mind the following best practices:

- Never specify a version string for a production application unless you are certain the application follows Oracle's programming conventions for production redeployment. See Developing Versioned Applications for Production Redeployment in *Developing Applications for Oracle WebLogic Server*.

- It is easiest to use production redeployment when applications are deployed in stage mode. With stage mode, WebLogic Server automatically creates a separate directory for each different version of the application that is deployed. These directories are stored in the configured staging directory for the server (by default, the *server_name*/stage subdirectory of the domain directory) and are removed when an the associated application version is undeployed.
- If you deploy in nostage mode, store each new version of an application in a dedicated subdirectory. This ensures that you do not overwrite older versions of your deployment files, and allows you to revert to an earlier application version if you detect problems after an update.
- If you deploy in external_stage mode, you must store the deployment files for each application version in the correct version subdirectory of each target server's staging directory. For example, the versioned application files used in the previous sample commands would need to be copied into the subdirectories `/myTestDeployment/.91Beta` and `/myTestDeployment/1.0GA` in each server's staging directory before deployment and redeployment.

Using In-Place Redeployment for Applications and Standalone Modules

In-place redeployment immediately undeploys and replaces a running application's deployment files with updated deployment files.

When used to redeploy entire applications or Jakarta EE modules, in-place redeployment makes the application or module unavailable during the deployment process, and can cause existing clients to lose in-process work. This disruption of client services occurs because application class files and libraries are immediately undeployed from their classloaders and then replaced with updated versions.

Because in-place redeployment of applications and modules adversely affects clients of the application, it should not be used with production applications unless:

- No clients are currently using the application, or
- It is acceptable to lose client access to the application and in-process work during redeployment.

> **Note:**
>
> WebLogic Server always performs in-place redeployment for applications that do not include a version identifier.

WebLogic Server also uses in-place redeployment for many partial redeployment operations (redeployment commands that affect only a portion of an application). In some cases, using partial redeployment is acceptable with production applications, because the redeployed files do not adversely affect active client connections. Table 8-1 describes each type of partial deployment and its effect on deployed applications.

Table 8-2 Partial Redeployment Behavior

Scope of Partial Redeployment	Redeployment Behavior	Recommended Usage
Graphics files, static HTML files, JSPs	Source files are immediately replaced on-disk and served on the next client request.	Use only during scheduled application downtime, or when it is not critical to preserve client connections and in-process work.
Jakarta EE Modules in an enterprise application	All files are immediately replaced. Java class files and libraries are unloaded from classloaders and replaced with updated files.	Use only during scheduled application downtime, or when it is not critical to preserve client connections and in-process work.
Deployment plan with dynamic property changes (such as tuning parameters)	The application is updated in-place. If the application is versioned, the plan version is *not* incremented.	Safe for all production environments.
Deployment plan with non-dynamic property (resource binding) changes	If the application is versioned, is compatible with production redeployment, and is redeployed, WebLogic Server increments the version identifier and uses the production redeployment strategy to update the application. If the application cannot use production redeployment, you must redeploy the entire application.	Safe for versioned applications that are compatible with production redeployment. See Using Production Redeployment to Update Applications. If the application cannot use production redeployment, update the deployment plan only during scheduled application downtime or when it is not critical to preserve client connections and in-process work. You must redeploy (instead of update) applications with deployment plans that contain changes to non-dynamic properties. Attempts to update applications with such plans will fail.

- Redeploying Applications and Modules In-Place
- Best Practices for Redeploying Applications and Modules In-Place

Redeploying Applications and Modules In-Place

To redeploy an entire application or standalone module using the in-place redeployment strategy:

1. If you want to preserve client connections to the application, first take the application offline and verify that no clients are accessing the application.

   The exact method for taking an application offline depends on the architecture of your WebLogic Server domain. In most cases, a redundant server or cluster is created to host a separate copy of the application, and load balancing hardware or software manages access to both servers or clusters. To take the application offline, the load balancing policies are changed to roll all client connections from one set of servers or clusters to the redundant set.

2. Place the new application deployment files in a suitable location for deployment. Oracle recommends that you store each version of an application's deployment files unique subdirectories.

   For example, if the currently deployed application's files (e.g., `SimpleEAR.ear`) are stored in:

   ```
   /myDeployments/myApplication/91Beta/SimpleEAR.ear
   ```

   You would store the updated application files in a new directory, such as:

```
/myDeployments/myApplication/1.0GA/SimpleEAR.ear
```

3. Redeploy the application and specify the updated deployment source files. To redeploy the application on all configured target servers, specify only the deployment name, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -redeploy -source /myDeployments/myApplication/1.0GA/
SimpleEAR.ear -name myApp
```

   If an application was previously deployed to multiple, non-clustered server instances, you can specify a target list to redeploy the application on only a subset of the target servers, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -redeploy -source /myDeployments/myApplication/1.0GA/
SimpleEAR.ear -name myApp
   -targets myserver1,myserver2
```

> **Note:**
>
> For applications deployed to a cluster, redeployment always occurs on all target server instances in the cluster. If the application was previously deployed to all servers in the cluster, you cannot subsequently redeploy the application on a subset of servers in the cluster.

4. If you took the server or cluster hosting the application offline, bring the host servers back online after the redeployment completes.
5. If necessary, restore the load balancing policies of your load balancing hardware or software to migrate clients from the temporary servers back to the online production servers.

Best Practices for Redeploying Applications and Modules In-Place

When using in-place redeployment to redeploy entire applications or standalone modules, keep in mind the following:

- Redeploying entire, staged applications may have performance implications due to increased network traffic when deployment files are copied to the Managed Servers. If you have very large applications, consider using the external_stage mode and copying deployment files manually before you redeploy the application. See Using External_stage Mode Deployment.
- Applications deployed to a WebLogic Server cluster must always be redeployed on all members of the cluster. You cannot redeploy a clustered application to a subset of the cluster.
- To successfully redeploy an enterprise application, all of the application's modules must redeploy successfully.
- By default, WebLogic Server destroys current user sessions when you redeploy a Web application. If you want to preserve Web application user sessions during redeployment, set `save-sessions-enabled` to "true" in the `container-descriptor` stanza of the `weblogic.xml` deployment descriptor file. Note, however, that the application still remains unavailable while in-place redeployment takes place.

Using Partial Redeployment for Jakarta EE Module Updates

The `weblogic.Deployer` utility uses a different command form if you want to redeploy individual modules of a deployed enterprise application. To redeploy a subset of the modules of an enterprise application, specify *`modulename@servername`* in the target server list to identify the modules you want to redeploy.

For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -redeploy -name myApp
   -targets mymodule1@myserver1,mymodule2@myserver2
```

> **Note:**
>
> The use of `-redeploy` *`module-uri`* is deprecated. Instead, use production redeployment or redeploy the module using the `-targets` *`module@target`* syntax.

If the application was previously deployed to a cluster, you must redeploy the module to the entire cluster, rather than a subset of servers. If you specify a subset of servers in the cluster, `weblogic.Deployer` responds with the error:

```
An attempt to add server target target_name to module module_name has been rejected.
This is because its parent cluster, cluster_name, is aso targeted by the module.
```


Restrictions for Updating Jakarta EE Modules in an EAR

The following restrictions apply to using partial redeployment for modules in an enterprise application:

- If redeploying a single Jakarta EE module in an enterprise application would affect other Jakarta EE modules loaded in the same classloader, `weblogic.Deployer` requires that you explicitly redeploy all of the affected modules. If you attempt to use partial redeployment with only a subset of the affected Jakarta EE modules, `weblogic.Deployer` displays the error:

  ```
  Exception:weblogic.management.ApplicationException: [J2EE:160076] You must include
  all of [module_list] in your files list to modify [module]
  ```

- Remember that if you change an application's deployment descriptor files, the container redeploys the entire application even if you attempt a partial redeployment.
- JAR files in `WEB-INF/lib` cannot be redeployed independently of the rest of the Web application. The container automatically redeploys the entire application, but maintains the state, when redeploying JAR files in `WEB-INF/lib`.

Best Practices for Updating Jakarta EE Modules in an EAR

Keep in mind these best practices when using partial redeployment for enterprise application modules:

- When you use partial redeployment to redeploy a Jakarta EE module in an enterprise application, all classes loaded in the classloader for the updated module are reloaded. You can define custom class loading hierarchies in the WebLogic Server deployment descriptor to minimize the impact of partial redeployment to other modules in the application. See WebLogic Server Application Classloading in *Developing Applications for Oracle WebLogic Server* for more information on class loading behavior.
- Classes in the `WEB-INF/classes` directory can no longer be redeployed independently of the rest of the Web application. You can deploy only the updated classes (rather than the entire `WEB-INF/classes` directory) by setting the Reload Period for the Web application. (See weblogic.xml Schema in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server* for more information.) See also, Changing Classes in a Running Program in *Developing Applications for Oracle WebLogic Server* and Using FastSwap Deployment to Minimize Redeployment.
- By default, WebLogic Server destroys current user sessions when you redeploy a Web application module. If you want to preserve Web application user sessions during redeployment, set `save-sessions-enabled` to "true" in the `container-descriptor` stanza of the `weblogic.xml` deployment descriptor file. Note, however, that the application still remains unavailable while in-place redeployment takes place.

Updating Static Files in a Deployed Application

In a production environment, you may occasionally need to refresh the static content of a Web application module—HTML files, image files, JSPs, and so forth—without redeploying the entire application. If you deployed a Web application or an enterprise application as an exploded archive directory, you can use the `weblogic.Deployer` utility to update one or more changed static files in-place.

To redeploy a single file associated within a deployment unit, specify the file name at the end of the redeploy command. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name myApp -redeploy myApp/copyright.html
```

Always specify the pathname to updated files relative to the root directory of the exploded archive directory. In the above example, the Web application is deployed as part of an enterprise application, so the module directory is specified (`myApp/copyright.html`).

If the Web application module had been deployed as a standalone module, rather than as part of an enterprise application, the file would have been specified alone (`copyright.html`).

You can also redeploy an entire directory of files by specifying a directory name instead of a single file. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name myApp -redeploy myApp/myjsps
```

In the above example, all files and subdirectories located in the `myjsps` subdirectory of the enterprise application are redeployed in-place.

Updating the Deployment Configuration for an Application

After you have deployed an application or standalone module, you can change the WebLogic Server deployment configuration using either the WebLogic Remote Console or `weblogic.Deployer`.

The WebLogic Remote Console enables you to interactively modify individual deployment configuration properties, while `weblogic.Deployer` only allows you to specify an updated deployment plan file to use for reconfiguring the application.

Modifying a Configuration Using the Remote Console

The WebLogic Remote Console enables you to reconfigure all deployment configuration properties for an application, including properties that were not included in the application's deployment plan. If an application was deployed with a deployment plan, the Console displays any deployment plan configuration properties in the plan. See Modify a Deployment Plan in the *Oracle WebLogic Remote Console Online Help*.

The remaining configuration pages for an application enable you to change other WebLogic Server configuration properties. The exact properties that are available for configuration depend on the type of application or Jakarta EE module that is deployed. These pages are available regardless of whether or not the application was deployed with a deployment plan.

Note that certain configuration changes are safe to apply to running production applications, while other changes require you to shut down and restart the application. See Understanding Redeployment Behavior for Deployment Configuration Changes.

How Configuration Changes Are Stored

When you use the WebLogic Remote Console to make changes to properties that were defined in a deployment plan, the Console generates a new deployment plan that containing variable definitions for the new properties you modified, as well as any existing variables defined in the plan. You can select where to save the new deployment plan.

Updating an Application to Use a Different Deployment Plan

The `weblogic.Deployer` utility enables you to update an application's deployment configuration by providing a new deployment plan to use with the application.

> **Note:**
>
> The updated deployment plan must be valid for the application's current target servers, or the configuration update will fail.

To reconfigure an application with a different (and valid) deployment plan, use the **`-update`** command and specify the new deployment plan file, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -update -name myTestDeployment
     -plan /myDeployments/myNewPlan.xml
```

> **Note:**
>
> Certain configuration changes are safe to apply to running production applications, while other changes require you to shut down and restart the application. See Understanding Redeployment Behavior for Deployment Configuration Changes.

Understanding Redeployment Behavior for Deployment Configuration Changes

> **Note:**
>
> The redeployment behavior in this section applies both to configuration changes made using the WebLogic Remote Console and via the `weblogic.Deployer` **`-update`** command.

When you change the deployment configuration for a deployed application, WebLogic Server applies the changes using a redeployment operation. The type of redeployment strategy used depends on the nature of configuration changes applied. If you modified the value of a resource binding property, the configuration update uses either the production redeployment strategy (if the application supports production redeployment), or the in-place redeployment strategy for the entire application. Performing in-place redeployment for an entire application or module is not recommended for production environments, because the application is first undeployed, causing connected clients to lose in-progress work. See Overview of Redeployment Strategies for more information.

9

Managing Deployed Applications

Learn how to perform common maintenance tasks on applications and modules that are currently deployed to a WebLogic Server domain.
This chapter includes the following sections:

- Taking a Production Application Offline
  WebLogic Server provides two different ways to take an application offline for testing or maintenance purposes.
- Undeploying Shared Libraries and Packages
  A shared Jakarta EE libraries or optional package cannot be undeployed until all applications that reference the library or package are first undeployed.
- Adding a New Module to a Deployed Enterprise Application
  WebLogic Server's module-level targeting support enables you to add and deploy a new enterprise application module without having to redeploy other modules that are already deployed.
- Enabling Parallel Deployment for Applications and Modules
  For use cases involving the deployment of multiple applications or the deployment of a single application with multiple modules, parallel deployment improves startup and post-running deployment time, including server startup.
- Changing the Order of Deployment at Server Startup
  By default, WebLogic Server deploys applications and resources in a certain order.
- Changing the Target List for an Existing Deployment
  After you deploy an application or standalone module in a WebLogic Server domain, you can change the target server list to add new WebLogic Server instances or to remove existing server instances.
- Removing Files from a Web Application Deployment
  If you deploy a Web application using an exploded archive directory, you can update the static contents of the Web application in one of two ways.
- Managing Long-Running Deployment Tasks
  The WebLogic Server deployment system automatically assigns a unique ID to deployment tasks so that you can track and manage their progress.
- On-Demand Deployment of Internal Applications
  There are many internal applications that are deployed during startup. These internal applications consume memory and require CPU time during deployment. This contributes to the WebLogic Server startup time and base memory footprint. Because many of these internal applications are not needed by every user, WebLogic Server has been modified to wait and deploy these applications on the first access (on-demand) instead of always deploying them during server startup. This reduces startup time and memory footprint.
- Using the ReadyApp Framework
  The ReadyApp framework advertises the `READY` or `NOT READY` state of an application that is registered with the framework. When the ReadyApp endpoint, `/ready`, is queried by a load balancer or other similar source that has that functionality, the framework prevents traffic from being routed to a WebLogic Server instance that is not fully functional.

Taking a Production Application Offline

WebLogic Server provides two different ways to take an application offline for testing or maintenance purposes.

See the following sections:

- Stopping an Application to Restrict Client Access
- Undeploying an Application or Module

Stopping an Application to Restrict Client Access

Stopping an Application to Restrict Client Access makes an application unavailable for processing client requests, but does not remove the deployment from the WebLogic Server domain. Stopping an application places the deployment in administration mode, which allows you to perform internal testing using a configured administration channel.

As described in Distributing Applications to a Production Environment, distributing an application and starting it in administration mode, restricts access to an application to a configured administration channel. You can also stop a running application to client requests and place it in administration mode. In a production environment, you may want to stop an application to confirm a reported problem, or to isolate the application from external client processing in order to perform scheduled maintenance.

Use the **`-stop`** command to place a running application into administration mode, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -stop -adminmode
```

By default, WebLogic Server immediately stops the application, without regard to pending HTTP sessions or in-process work. If you want to wait for pending HTTP sessions to complete their work before stopping the application to client requests and placing it in administration mode, add the `-graceful` option:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -stop -adminmode -graceful
```

> **Note:**
>
> If you do not explicitly specify an application version with the `-appversion` option, the `-stop` command will only stop the active version of the application. If there are other versions of the application that you also want to stop (or that you want to stop instead or the active version), you must specify them with the `-appversion` option.

To restart an application that was previously stopped, making it available to external clients, use the **`-start`** command and specify the deployment name. You do not need to redeploy a stopped application to make it generally available:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -start
```

For detailed command information, see Start.

Undeploying an Application or Module

Undeploying an Application or Module makes an application unavailable for processing client requests and removes WebLogic Server-generated deployment files from the domain.

After you deploy a new application or standalone module to servers in a domain, the deployment name remains associated with the deployment files you selected. Even after you stop the deployment on all servers, the files remain available for redeployment using either the WebLogic Remote Console or `weblogic.Deployer` utility.

If you want to remove a deployment name and its associated deployment files from the domain, you must explicitly undeploy the application or standalone module. To undeploy a deployment unit from the domain using `weblogic.Deployer`, specify the `-undeploy` command, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -undeploy
```

> **Note:**
>
> Using the `-undeploy` command without the `-targets` and `-submoduletargets` flags completely removes the application or standalone module from all WebLogic Server instances and untargets all JMS sub-module resources.

By default, WebLogic Server immediately stops and undeploys the application, interrupting current clients and in-progress work. For a production application, you may want to undeploy "gracefully," allowing current HTTP clients to complete their work before undeploying:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -undeploy -graceful
```

Undeploying a deployment unit does not remove the original source files used for deployment. It only removes the deployment's configuration from the domain, as well as any deployment files that WebLogic Server created during deployment (for example, files copied with stage deployment mode and files uploaded to the Administration Server).

If you need to redeploy a deployment unit after undeploying it, you must again identify the deployment files, targets, staging mode, and deployment name using the instructions in Deploying Applications and Modules with weblogic.Deployer.

Undeploying Shared Libraries and Packages

A shared Jakarta EE libraries or optional package cannot be undeployed until all applications that reference the library or package are first undeployed.

If no applications reference an application or package, use the instructions in Undeploying an Application or Module to undeploy it.

Adding a New Module to a Deployed Enterprise Application

WebLogic Server's module-level targeting support enables you to add and deploy a new enterprise application module without having to redeploy other modules that are already deployed.

To deploy a new module in an EAR, you simply use module-level targeting syntax described in Module-Targeting Syntax.

For example, if you were to add a module, `newmodule.war`, to a deployed enterprise application named `myapp.ear` (update `application.xml` file as necessary), you could then deploy `newmodule.war` using the `weblogic.Deployer` command:

```
java weblogic.Deployer -username myname -password password
   -name myapp.ear -deploy -targets newmodule.war@myserver
   -source /myapp/myapp.ear
```

This command deploys the new module without redeploying the other modules in the application. Note that you must specify the correct file extension (`.war` in the above example) for archived modules in an EAR file.

Enabling Parallel Deployment for Applications and Modules

For use cases involving the deployment of multiple applications or the deployment of a single application with multiple modules, parallel deployment improves startup and post-running deployment time, including server startup.

When applications and resources are deployed, they are first sorted into sets by overall deployment type. Then, each set begins the deployment process, following the order described in Changing the Order of Deployment at Server Startup.

Within the set of applications, applications with the same deployment order number can deploy in parallel to each other, improving performance.

> **Note:**
>
> By default, parallel deployment for applications is enabled in WebLogic domains that are created with, or upgraded to, WebLogic Server 12.2.1 or later. (Parallel deployment for modules is disabled by default.)
>
> Applications running on WebLogic Server instances prior to version 12.2.1 may have some internal dependencies that must be specified in a deployment order. If that deployment order is not specified, then WebLogic Server cannot detect those dependencies. These dependencies would have been satisfied when using serial deployment on WebLogic Server versions prior to 12.2.1. However, there is some risk of these dependencies being violated when you make the transition from serial deployment to parallel deployment, and this violation would result in a deployment failure. Consequently, in domains that are created with versions of WebLogic Server prior to 12.2.1, parallel deployment is disabled by default. See Parallel Deployment in *Upgrading Oracle WebLogic Server*.

The following sections describe how to enable parallel deployment for applications and modules within an application archive:

- Enabling Parallel Deployment for Applications
- Enabling Parallel Deployment for Modules

Enabling Parallel Deployment for Applications

To enable parallel deployment for applications, configure the `ParallelDeployApplications` attribute of the `DomainMBean`. With this attribute enabled, applications with the same deployment order will deploy in parallel of each other, improving performance. The `ParallelDeployApplications` attribute is enabled by default.

The overall deployment order of resources and applications remains unchanged, so deployment types that precede applications in the normal deployment order (for example, JDBC system resources and Jakarta EE libraries) are guaranteed to be fully deployed before applications are deployed.

> **Note:**
>
> Only applications with no cross-dependencies should deploy in parallel. If an application depends on other applications, then this application needs to have a higher dependency-order than the applications it depends on. Otherwise, parallel activation may cause dependency failures when the dependent application attempts to deploy prior to the deployment of applications on which it depends.

Enabling Parallel Deployment for Modules

You can configure modules within a single application archive to deploy in parallel to each other. To enable parallel deployment of modules for all applications in the domain, configure the `ParallelDeployApplicationModules` attribute of the `DomainMBean`. The `ParallelDeployApplicationModules` attribute is disabled by default.

To enable parallel deployment of modules on a per-application basis, configure the `ParallelDeployModules` attribute of the `AppDeploymentMBean`. Setting this attribute overrides the domain value for an individual application.

> **Note:**
>
> You can only enable parallel deployment of modules for applications that contain modules with no cross-dependencies. If modules are dependent on other modules, parallel deployment may cause dependency failures when the dependent module attempts to deploy prior to the deployment of the modules on which it depends.

> **Note:**
>
> If the Jakarta EE `<initialize-in-order>` element is specified and set to `true` for an application, then the modules within the application are deployed according to the order they are defined in `application.xml`, regardless of whether parallel deployment of modules is enabled or disabled.

Changing the Order of Deployment at Server Startup

By default, WebLogic Server deploys applications and resources in a certain order.

1. Caches
2. Internal applications
3. JDBC system resources
4. Deployment handlers
5. JMS system resources
6. Resource-dependent deployment handlers
7. Startup classes
8. WLDF system resources
9. Jakarta EE libraries and optional packages
10. Applications and standalone modules
11. Coherence cluster system resources
12. Custom system resources (this module type is mapped to the `custom-resource` deployment descriptor (DD) element)

> **Note:**
>
> WebLogic Server security services are always initialized before server resources, applications, and startup classes are deployed. For this reason, you cannot configure custom security providers using startup classes, nor can custom security provider implementations rely on deployed server resources such as JDBC.

- Changing the Deployment Order for Applications and Standalone Modules
- Changing the Deployment Order for Modules in an Enterprise Application
- Ordering Startup Class Running and Deployment

Changing the Deployment Order for Applications and Standalone Modules

You can change the deployment order for a deployed application or *standalone* module by setting the `AppDeploymentMBean DeploymentOrder` attribute in the WebLogic Remote Console (or programmatically using the `AppDeploymentMBean`). The `DeploymentOrder` attribute controls the load order of deployments relative to one another—modules with lower `DeploymentOrder` values deploy before those with higher values. By default, each deployment unit is configured with a Deployment Order value of 100. Deployments with the same Deployment Order value are deployed in alphabetical order using the deployment name. In all cases, applications and standalone modules are deployed after the WebLogic Server instance has initialized dependent subsystems.

> **Note:**
>
> You cannot change the load order of applications and standalone modules using the weblogic.Deployer utility.

Using the WebLogic Remote Console, view or change the deployment order, in the **Edit Tree**, on the **Deployments**: App **Deployments**: *myAppDeployment*: **Overview** page.

Changing the Deployment Order for Modules in an Enterprise Application

The modules contained in an enterprise application are deployed in the order in which they are declared in the `application.xml` deployment descriptor. See Enterprise Application Deployment Descriptor Elements in *Developing Applications for Oracle WebLogic Server*.

Ordering Startup Class Running and Deployment

By default WebLogic Server startup classes are run after the server initializes JMS and JDBC services, and after applications and standalone modules have been deployed.

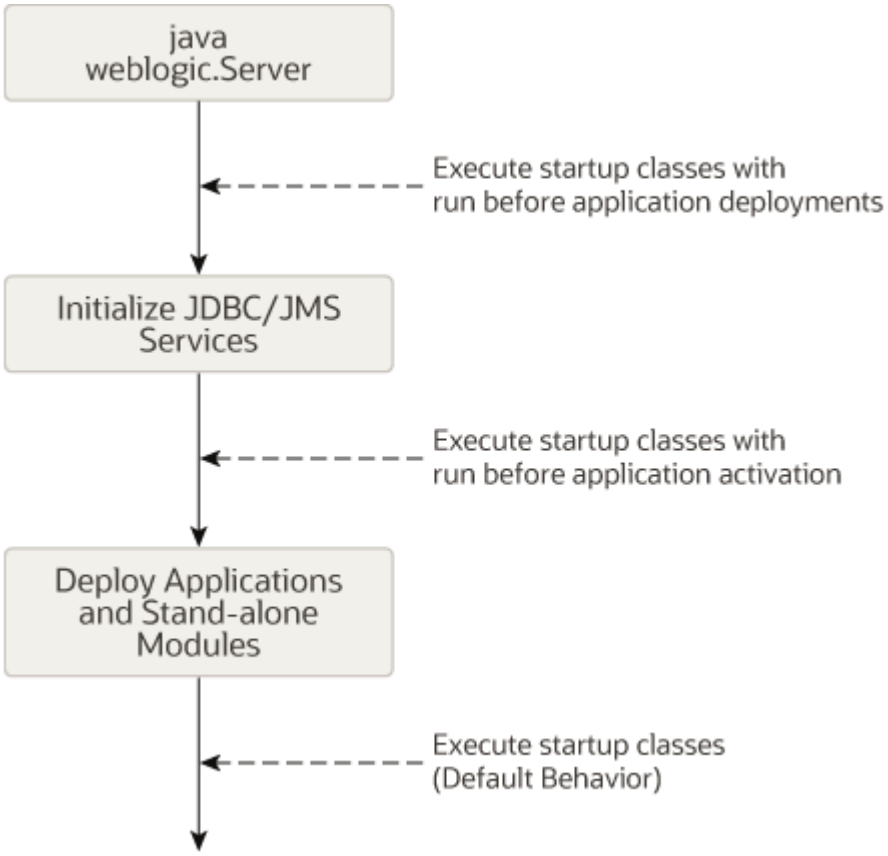
If you want to perform startup tasks after JMS and JDBC services are available, but before applications and modules have been activated, you can select the **Run Before Application Deployments** option in the WebLogic Remote Console (or set the `StartupClassMBean`'s `LoadBeforeAppActivation` attribute to `true`).

If you want to perform startup tasks before JMS and JDBC services are available, you can select the **Run Before Application Activations** option in the WebLogic Remote Console (or set the `StartupClassMBean`'s `LoadBeforeAppDeployments` attribute to `true`).

To select **Run Before Application Deployments** or **Run Before Application Activations** in the WebLogic Remote Console, see Configure Startup Classes in the *Oracle WebLogic Remote Console Online Help*.

The following figure summarizes the time at which WebLogic Server runs startup classes.

Figure 9-1 Startup Class Execution



For more information, see the description of the Javadocs for `StartupClassMBean`.

Changing the Target List for an Existing Deployment

After you deploy an application or standalone module in a WebLogic Server domain, you can change the target server list to add new WebLogic Server instances or to remove existing server instances.

If you remove a target server, only the target list itself is updated—the deployment unit remains deployed to the removed server until you explicitly undeploy it. Similarly, if you add a new target server, you must explicitly deploy the deployment unit on the new server before it is active on that server.

To add a new server to the target list using `weblogic.Deployer`, simply specify the new list of target servers with the **`-deploy`** command. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mydeploymentname -deploy
   -targets server1, newserver
```

Removing Files from a Web Application Deployment

If you deploy a Web application using an exploded archive directory, you can update the static contents of the Web application in one of two ways.

You can refresh the files (see Updating Static Files in a Deployed Application), or delete the files from the deployment. To delete files, you must use the `weblogic.Deployer` utility with the `delete_files` option. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mywebapp -delete_files mywebapp/copyright.html
```

Always specify the pathname to updated files starting with the top level of the exploded archive directory. In the above example, the Web application resides in an exploded archive directory named `mywebapp`.

> **Note:**
>
> Because the `-delete_files` option deletes all specified files or, if you specify a directory but do not specify files within the directory, all files in the specified directory, Oracle recommends that you use caution when using the delete_files option and that you do not use the `delete_files` option in production environments.

Managing Long-Running Deployment Tasks

The WebLogic Server deployment system automatically assigns a unique ID to deployment tasks so that you can track and manage their progress.

`weblogic.Deployer` enables you to assign your own task identification numbers for use with deployment commands, as well as monitor and cancel long-running deployment tasks.

For example, the following command assigns a task ID of `redeployPatch2` to a new deployment operation:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -targets myserver -id redeployPatch2
   -nowait -deploy c:\localfiles\myapp.ear
```

You can later check the status of the task:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -id redeployPatch2 -list
```

You can check the status of all running tasks:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -listtask
```

If a task takes too long to complete you can cancel it:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -id redeployPatch2 -cancel
```

On-Demand Deployment of Internal Applications

There are many internal applications that are deployed during startup. These internal applications consume memory and require CPU time during deployment. This contributes to the WebLogic Server startup time and base memory footprint. Because many of these internal applications are not needed by every user, WebLogic Server has been modified to wait and

deploy these applications on the first access (on-demand) instead of always deploying them during server startup. This reduces startup time and memory footprint.

The following sections describe on-demand deployment in detail.

- Internal Application Types
- Configuring On-Demand Deployment

Internal Application Types

There are two different types of internal applications. The first type displays a user interface, and includes the WebLogic Remote Console, UDDI explorer, and WebLogic Server test client. The second type does not display a user interface, and includes UDDI and internal servlets for deployment and management file distribution.

For applications with a user interface, WebLogic Server displays a status page indicating that on-demand deployment is in progress. This page refreshes every two seconds. When the internal application completes deployment, you are redirected to the internal application. This status page is displayed on the first access of each application. Subsequent invocations do not deploy the application and go directly to the user interface for the internal application.

Configuring On-Demand Deployment

For a development domain, the default is for WebLogic Server to deploy internal applications on demand. For a production-mode domain, the default is for WebLogic Server to deploy internal applications as part of server startup. You can control the default behavior by configuring the `InternalAppsDeployOnDemandEnabled` attribute in the Domain MBean. You can change the configuration setting using the WebLogic Remote Console or by using the WebLogic Scripting Tool (WLST).

- Configuring On-Demand Deployment Using the Remote Console
- Configuring On-Demand Deployment Using WLST

Configuring On-Demand Deployment Using the Remote Console

To change the `InternalAppsDeployOnDemandEnabled` attribute using the WebLogic Remote Console, perform the following steps:

1. In the **Edit Tree**, go to the **Environment**: **Domain**: **General** page.
2. Change the setting of the **Enable on-demand deployment of internal applications** option.
3. Click **Save**.
4. Open the Shopping Cart and then click **Commit Changes**.

Configuring On-Demand Deployment Using WLST

To change the `InternalAppsDeployOnDemandEnabled` attribute using the WLST, perform the following steps:

```
connect()
edit()
startEdit()
cmo.setInternalAppsDeployOnDemandEnabled(false)
activate()
```

Using the ReadyApp Framework

The ReadyApp framework advertises the `READY` or `NOT READY` state of an application that is registered with the framework. When the ReadyApp endpoint, `/ready`, is queried by a load balancer or other similar source that has that functionality, the framework prevents traffic from being routed to a WebLogic Server instance that is not fully functional.

When applications that are deployed to WebLogic Server experience asynchronous behavior during initialization, WebLogic Server may enter the `RUNNING` state before the applications finish initializing. In this situation, the load balancer or other instances of WebLogic Server may prematurely route traffic to a server that is not fully functional. If you rely on the server to determine whether applications are ready to service requests, errors may result.

The ReadyApp framework introduces the `READY` and `NOT READY` states, which allow internal and external applications to influence the state of the server in which the application runs.

For example, if an application that is registered with the ReadyApp framework is not ready to service requests from an instance of WebLogic Server in the `RUNNING` state, the sources that query the `/ready` endpoint internally mark the state of the server instance and application as unavailable until the application calls the `ready()` method.

- Configuring the ReadyApp Framework with EAR-based or WAR-based Applications
- Monitoring the ReadyApp Framework

Configuring the ReadyApp Framework with EAR-based or WAR-based Applications

To use the ReadyApp framework, register the EAR-based or WAR-based application with the framework by adding the following code to the application's WebLogic deployment descriptor (`META-INF\weblogic-application.xml`):

```
<wls:ready-registration>true</wls:ready-registration>
```

When the application starts, the state is set to `NOT READY`. Any application that is registered with the ReadyApp framework is automatically unregistered when the application is undeployed from WebLogic Server.

> **Note:**
>
> The prefix `wls:` may not be required, depending on the contents of the `weblogic-application.xml` file. If the rest of the tags do not have the prefix, you can ignore the prefix.

When the application has finished initializing and is ready to service requests, set the application to the `READY` state by adding the following line to the code where the initialization completes:

```
weblogic.application.ready.ReadyLifecycleManager.getInstance().ready();
```

A call to the `ready()` method indicates to the ReadyApp framework that the application is ready to process requests. However, the application `READY` state does not mean that the server

is ready. All other registered applications must signal that they are ready before the server is ready.

If the application has to stop processing requests, (for example, if the application needs to be reinitialized), use the following method to indicate that the application is not ready:

```
weblogic.application.ready.ReadyLifecycleManager.getInstance().notReady();
```

> **Note:**
>
> After an application calls the `notReady()` method, the server where the application is deployed remains in the `NOT READY` state until the application calls the `ready()` method again.

The `ready()` and `notReady()` methods can cause the following runtime exceptions:

- `IllegalArgumentException`: This exception occurs when the `applicationId` reported by the Component Invocation Context is null.
- `IllegalStateException`: This exception occurs when the application has not been properly registered. Check the deployment descriptor for proper setup.

Monitoring the ReadyApp Framework

To check the running status of the ReadyApp framework, use one of the following methods:

- Visit the ReadyApp servlet URL.
- Enable the `DebugReadyApp` option in the WebLogic Remote Console.

- Monitoring the ReadyApp Framework Using the ReadyApp Servlet URL
- Monitoring the ReadyApp Framework by Enabling DebugReadyApp in the WebLogic Remote Console

Monitoring the ReadyApp Framework Using the ReadyApp Servlet URL

Point your browser to `http://{host:port}/weblogic/ready`. The browser returns a page with either a status 200 (`READY`) or 503 (`NOT READY`). Be sure to replace *`host`* and *`port`* with the host name and port number of the desired server. When WebLogic Server is not running, an Error 404 page appears.

The `http://{host:port}/weblogic/ready` URL opens a blank page with only the HTTP status set. Use a browser tool to see the actual status. For example, the Google Chrome browser has a Developer Tools setting to show the status of the page and the latency.

Monitoring the ReadyApp Framework by Enabling DebugReadyApp in the WebLogic Remote Console

You can also monitor the ReadyApp framework by enabling the `DebugReadyApp` option in the WebLogic Remote Console. If `STDOUT` logging is enabled, this method of monitoring writes information to the log file or the Console.

To monitor the ReadyApp framework using the WebLogic Remote Console:

1. In the **Edit Tree**, go to **Environment**: **Servers**: *myServer*.

2. Then, go to the **Debug**: **Application** page.
3. Select the **DebugReadyApp** check box, and then click **Save**.

   This change does not require WebLogic Server to restart.
4. Set the logging severity level to Debug:
   a. In the **Edit Tree**, go to **Environment**: **Servers**: *myServer*.
   b. Then, select the **Logging** tab, the **General** subtab, and click **Show Advanced Fields**
   c. Set the severity level of **Minimum severity to log** to Debug.
   d. Set the **Log File Severity Level** to Debug.
   e. Click **Save**.
5. Visit `http://{host:port}/weblogic/ready`. Output similar to Example 9-1 appears:

Example 9-1 The ReadyApp Framework Status

```
**********************
Ready App Map - Operation: Get Ready Status
Partition Id GLOBAL
Item 0 key: TestEar value: 1
Partition Id ratestp2
Item 0 key: ReadyApp2Test$ratestp2 value: 1
Partition Id ratestp1
Item 0 key: ReadyApp2Test$ratestp1 value: 1
**********************
```

In Example 9-1, three applications are deployed to the server instance: one in the `GLOBAL` partition and two in the `ratest` partitions. A value of `1` indicates that these three applications are not ready. If the `ready()` method is called on these applications, the value is set to `0`, which indicates the `READY` state.

> **Note:**
>
> WebLogic Server Multitenant domain partitions were deprecated in WebLogic Server 12.2.1.4.0 and are removed in this release.

A

weblogic.Deployer Command-Line Reference

Learn about the `weblogic.Deployer` utility, a Java-based deployment tool that provides WebLogic Server administrators and developers command-line based deployment operations.

> **Note:**
>
> See the *WLST Command Reference for Oracle WebLogic Server* for information about performing deployment operations using the WebLogic Scripting Tool (WLST).

This appendix includes the following sections:

- Required Environment for weblogic.Deployer
  Set up your environment to use the `weblogic.Deployer` utility.
- Syntax for Invoking weblogic.Deployer
- Commands and Options
  Examine the `weblogic.Deployer` commands and command options used to perform deployment tasks with WebLogic Server.
- Exit Codes
  An exit code of zero (0) indicates success, whereas, a non-zero exit code indicates one or more failures were detected, the value being the number of failures.
- Example config.xml File and Corresponding weblogic.Deployer Command
  Examine an application's `config.xml` file and the corresponding `weblogic.Deployer` command to deploy the application.

Required Environment for weblogic.Deployer

Set up your environment to use the `weblogic.Deployer` utility.

1. Install and configure the WebLogic Server software, as described in the *Installing and Configuring Oracle WebLogic Server and Coherence*.
2. Add the WebLogic Server classes to the `CLASSPATH` environment variable, and ensure that the correct JDK binaries are available in your `PATH`. 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 environment.

> **Note:**
>
> On UNIX operating systems, the `setWLSEnv.sh` command does not set the environment variables in all command shells. Oracle recommends that you execute this command using the Korn shell or bash shell.

3. If you are connecting to an Administration Server through a configured administration channel, you must also configure SSL on the machine on which you run `weblogic.Deployer`. See Configuring SSL in *Administering Security for Oracle WebLogic Server* for instructions about configuring SSL.

Syntax for Invoking weblogic.Deployer

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

Command names and options are not case-sensitive. See Commands and Options for detailed syntax and examples of using `weblogic.Deployer` commands.

> **Note:**
>
> Entering unencrypted passwords in `weblogic.Deployer` commands, whether entered in a command window or a script, is a security risk because they can be viewed by others. For information about entering passwords securely, see Do Not Include Unencrypted Passwords in Commands and Scripts in *Securing a Production Environment for Oracle WebLogic Server*.

- SSL Arguments
- Connection Arguments
- User Credentials Arguments
- Common Arguments

SSL Arguments

```
java [ -Dweblogic.security.TrustKeyStore=DemoTrust ]
     [ -Dweblogic.security.JavaStandardTrustKeyStorePassPhrase=password ]
     [ -Dweblogic.security.CustomTrustKeyStoreFileName=filename
       -Dweblogic.security.TrustKeystoreType=jks
       [-Dweblogic.security.CustomTrustKeyStorePassPhrase=password ]
     ]
     [ -Dweblogic.security.SSL.hostnameVerifier=classname ]
     [ -Dweblogic.security.SSL.ignoreHostnameVerification=true ]
weblogic.Deployer
     [ User Credentials Arguments ]
     COMMAND-NAME command-arguments
```

If you have enabled the domain-wide administration port, or if you want to secure your administrative request by using some other listen port that is secured by SSL, you must include SSL arguments when you invoke `weblogic.Deployer`. Table A-1 describes all SSL arguments for the `weblogic.Deployer` utility.

Table A-1 SSL Arguments

Argument	Definition
`-Dweblogic.security.TrustKeyStore=DemoTrust`	Causes `weblogic.Deployer` to trust the CA certificates in the demonstration trust keystore (*`WL_HOME`*`\server\lib\DemoTrust.jks`). This argument is required if the server instance to which you want to connect is using the demonstration identity and certificates. By default, `weblogic.Deployer` trusts only the CA certificates in the Java Standard Trust keystore (*`SDK_HOME`*`\jre\lib\security\cacerts`).
`-Dweblogic.security.JavaStandardTrustKeyStorePassPhrase=`*`password`*	Password that was used to secure the Java Standard Trust keystore. If the Java Standard Trust keystore is protected by a password, and if you want to trust its CA certificates, you must use this argument. By default, the Java Standard Trust keystore is not protected by a password.
`-Dweblogic.security.CustomTrustKeyStoreFileName=`*`filename`* `-Dweblogic.security.TrustKeystoreType=jks`	Causes `weblogic.Deployer` to trust the CA certificates in a custom keystore that is located at *filename*. You must use both arguments to trust custom keystores. The *filename* must match exactly the `ServerMBean.CustomTrustKeyStoreFileName` value persisted in `config.xml`; if the value specified in the `CustomTrustKeyStoreFileName` attribute is a relative pathname, you must also specify the same relative pathname in this argument.
`-Dweblogic.security.CustomTrustKeyStorePassPhrase=`*`password`*	Password that was used to secure the custom keystore. You must use this argument only if the custom keystore is protected by a password.
`-Dweblogic.security.SSL.hostnameVerifier=`*`classname`*	Name of a custom Host Name Verifier class. The class must implement the `weblogic.security.SSL.HostnameVerifier` interface.
`-Dweblogic.security.SSL.ignoreHostnameVerification=true`	Disables host name verification.
`-Dweblogic.RootDirectory=`*`DOMAIN_HOME`*	Specifies the server's root directory. See A Server's Root Directory in *Understanding Domain Configuration for Oracle WebLogic Server*. **Note**: This argument is required if your domain has SSL configured and you are using the demo trust keystore.

Connection Arguments

```
java [SSL Arguments] weblogic.Deployer
     [-adminurl protocol://listen_address:port_number]
     [User Credentials Arguments] COMMAND-NAME command-options [Common Arguments]
```

Most `weblogic.Deployer` commands require you to specify the `-adminurl` arguments described in Table A-2 to connect to an Administration Server instance.

Table A-2 Connection Arguments

Argument	Definition
`-adminurl [protocol://] Admin-Server-listen-address:listen-port`	Listen address and listen port of the Administration Server. To use a port that is not secured by SSL, the format is `-adminurl [protocol]Admin-Server-listen-address:port` where `t3`, `http`, `iiop`, and `iiops` are valid protocols. In order to use an adminurl with the HTTP protocol, you must enable the HTTP tunneling option in the WebLogic Remote Console. See Setting Up WebLogic Server for HTTP Tunneling in *Administering Server Environments for Oracle WebLogic Server* and Configure Network Connections in the *Oracle WebLogic Remote Console Online Help*. To use a port that is secured by SSL, the format is `-adminurl secure-protocol://Admin-Server-listen-address:port` where `t3s` and `https` are valid secure protocols. To connect to the Administration Server via a configured administration channel, you must specify a valid administration port number: `-adminurl secure-protocol://Admin-Server-listen-address:domain-wide-admin-port` There is no default value for this argument.

User Credentials Arguments

```
java [ SSL Arguments ] weblogic.Deployer [Connection Arguments]
     [ { -username username [-password password] } |
     [ -userconfigfile config-file [-userkeyfile admin-key] ] ]
     COMMAND-NAME command-options [Common Arguments]
```

Most `weblogic.Deployer` commands require you to provide the user credentials of a WebLogic Server administrator.

Table A-3 User Credentials Arguments

Argument	Definition
`-username username`	Administrator username. If you supply the `-username` option but you do not supply a corresponding `-password` option, `weblogic.Deployer` prompts you for the password.
`-password password`	Password of the administrator user. To avoid having the plain text password appear in scripts or in process utilities such as `ps`, first store the username and encrypted password in a configuration file using the WebLogic Scripting Tool (WLST) `storeUserConfig` command as described in the *WLST Command Reference for WebLogic Server*. Omit both the `-username` and `-password` options to `weblogic.Deployer` to use the values stored in the default configuration file. If you want to use a specific configuration file and key file, rather than the default files, use the `-userconfigfile` and `-userkeyfile` options to `weblogic.Deployer`.

Table A-3 (Cont.) User Credentials Arguments

Argument	Definition
`-userconfigfile` *`config-file`*	Location of a user configuration file to use for the administrative username and password. Use this option, instead of the `-user` and `-password` options, in automated scripts or in situations where you do not want to have the password shown on-screen or in process-level utilities such as `ps`. Before specifying the `-userconfigfile` attribute, you must first generate the file using the WebLogic Scripting Tool (WLST) `storeUserConfig` command as described in the *WLST Command Reference for WebLogic Server*.
`-userkeyfile` *`admin-key`*	Specifies the location of a user key file to use for encrypting and decrypting the username and password information stored in a user configuration file (the `-userconfigfile` option). Before specifying the `-userkeyfile` attribute, you must first generate the file using the WebLogic Scripting Tool (WLST) `storeUserConfig` command as described in the *WLST Command Reference for WebLogic Server*.

Common Arguments

```
java [
          SSL Arguments
       ] weblogic.Deployer [Connection Arguments]
    [ { -username username [-password password] } |
    [ -userconfigfile config-file [-userkeyfile admin-key] ] ]
    COMMAND-NAME
       command-options [Common Arguments]
```

The common arguments described in Table A-4 can be used with any commands described in Commands and Options.

Table A-4 Common Arguments for weblogic.Deployer

Argument Name	Description
`-advanced`	Prints full command-line help text for all `weblogic.Deployer` actions and options.
`-debug`	Display debug messages in the standard output.
`-examples`	Display example command lines for common tasks.
`-help`	Prints command-line help text for the most commonly used `weblogic.Deployer` actions and options.
`-noexit`	By default `weblogic.Deployer` calls `System.exit(1)` if an exception is raised during command processing. The exit value displayed indicates the number of failures that occurred during the deployment operation. The `-noexit` option overrides this behavior for batch processing.
`-noversion`	Indicates that `weblogic.Deployer` should ignore all version related code paths on the Administration Server. This behavior is useful when deployment source files are located on Managed Servers (not the Administration Server) and you want to use the external_stage staging mode. If you use this option, you cannot use versioned applications.
`-nowait`	`weblogic.Deployer` prints the task ID and exits without waiting for the action to complete. This option initiates multiple tasks and then monitor them later with the `-list` action.

Table A-4 (Cont.) Common Arguments for weblogic.Deployer

Argument Name	Description
`-output <raw | formatted>`	(Deprecated.) Either `raw` or `formatted` to control the appearance of `weblogic.Deployer` output messages. Both output types contain the same information, but `raw` output does not contain embedded tabs. By default, `weblogic.Deployer` displays `raw` output.
`-remote`	Indicates that `weblogic.Deployer` is not running on the same machine as the Administration Server, and that source paths specified in the command are valid for the Administration Server machine itself. If you do not use the `-remote` option, `weblogic.Deployer` assumes that all source paths are valid paths on the local machine. If `-upload` is specified and the admin URL is t3 over SSL/TLS(t3s://...), the Deployer will open a HTTPS connection to upload the file being deployed. By default, the JDK's HTTP client is used. A proper trust store needs to be configured with `-Djavax.net.ssl.trustStore=path_to_trust_store` and `-Djavax.net.ssl.trustStorePassword=password_of_trust_store`. To use WebLogic's HTTP client for the file upload, which shares the trust store used by the t3s connection, set `-Djava.protocol.handler.pkgs=weblogic.net`.
`-timeout seconds`	Maximum time, in seconds, to wait for the deployment task to complete. After the time expires, `weblogic.Deployer` prints out the current status of the deployment and exits.
`-verbose`	Additional progress messages, including details about the prepare and activate phases of the deployment.
`-version`	Prints version information for `weblogic.Deployer`.

Commands and Options

Examine the `weblogic.Deployer` commands and command options used to perform deployment tasks with WebLogic Server.

- Cancel
- Deploy
- Distribute
- Listapps
- List, Listtask
- Purgetasks
- Redeploy
- Start
- Stop
- Undeploy
- Update

Cancel

Attempt to cancel a running deployment task.

- Syntax
- Examples

Syntax

```
java [SSL Arguments] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -cancel task_id
     [Common Arguments]
```

Command or Option	Definition
`task_id`	Identifier of the deployment task to cancel. Specify the identifier by using the `id` option with the deploy, distribute, update, undeploy, redeploy, stop, and start commands.

Examples

The following command starts a deployment operation and specifies the task identifier, `myDeployment`:

```
java weblogic.Deployer -adminurl http://localhost:7001
     -username weblogic -password weblogic
     -deploy ./myapp.ear -id myDeployment
```

If the deployment task has not yet completed, the following command attempts to cancel the deployment operation:

```
java weblogic.Deployer -adminurl http://localhost:7001
     -username weblogic -password weblogic
     -cancel -id myDeployment
```

Deploy

Deploys or redeploys an application or module.

> **Note:**
>
> The `-ACTIVATE` command, an alias for deploy, is deprecated.

- Syntax
- Examples

Syntax

```
java [SSL Arguments] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -deploy [[-name] deployment_name] [-source] file
     [-editsession edit_session_name]
     [-plan file] [-targets target_list] [-submoduletargets target_list]
     [-upload]
     [-stage | -nostage | -external_stage]
     [-planstage | -plannostage | -planexternal_stage]
```

```
[-retiretimeout seconds]
[-library [-libspecver version] [-libimplver version]]
[-altappdd file] [-altwlsappdd file]
[-securityModel] [-enableSecurityValidation]
[-id task_id]
[-extendLoader]
[-dbClientData] [-dbClientDataUploadPath]
[Common Arguments]
```

Command or Option	Definition
`-name deployment_name`	Deployment name to assign to a newly-deployed application or standalone module. Both the `-name` option and *`deployment_name`* argument are optional, as described in the Syntax. If a deployment name is not explicitly identified with the **deploy** command, the name is derived from the specified deployment file or directory: • For an archive file, the default deployment name is the full name of the archive file with the file extension. For example, when deploying `myear.ear`, the default deployment name is `myear.ear`. • For an exploded archive directory, the default deployment name is the name of the top-level directory. • If you specify an application installation root directory, the default deployment name is derived from the archive filename or exploded archive directory name in the `/app` subdirectory.
`-source file`	Archive file or exploded archive directory to deploy. You can omit the `-source` option and supply only the file or directory to deploy.
`-editsession edit_session_name`	Specifies the name of the edit session that was active during this deployment operation. If an edit session is specified, then the deployment will be associated with the edit session and deferred until the edit session is activated.
`-plan file`	Deployment plan to use when deploying the application or module. By default, `weblogic.Deployer` does not use an available deployment plan, even if you are deploying from an application root directory that contains a plan.
`-targets target_list`	Targets on which to deploy the application or module. The *`target_list`* argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a Java EE module name (`<module1>@<server1>`). This enables you to deploy different modules of an enterprise application to different servers or clusters. See Using Module-Level Targeting for Deploying an Enterprise Application. If you do not specify a target list with the **deploy** command, the target defaults to: • The Administration Server instance for new deployments. • The application's current targets for deployed applications.
`-submoduletargets target_list`	JMS Server targets for resources defined within a JMS application module. See Using Sub-Module Targeting with JMS Application Modules and Using WLST to Manage JMS Servers and JMS System Resources in *Administering JMS Resources for Oracle WebLogic Server*.
`-upload`	Transfers the specified deployment files, including deployment plans and alternate deployment descriptors, to the Administration Server. Use this option when you are on a remote machine and you cannot copy the deployment files to the Administration Server by other means. The application files are uploaded to the WebLogic Server Administration Server upload directory prior to distribution and deployment.

Command or Option	Definition
`-stage | -nostage | -external_stage`	Staging mode to use when deploying or distributing an application: • `-stage`—Copies deployment files to target servers' staging directories. `stage` is the default mode used when deploying or distributing to Managed Server targets. • `-nostage`—Does not copy the deployment files to target servers, but leaves them in a fixed location, specified by the `-source` option. Target servers access the same copy of the deployment files. `nostage` is the default used when deploying or distributing to the Administration Server (for example, in a single-server domain). • `-external_stage`—Does not copy the deployment files to target servers; instead, you must ensure that deployment files have been copied to the correct subdirectory in the target servers' staging directories. You can manually copy the files or use a third-party tool or script. See Controlling Deployment File Copying with Staging Modes.
`-planstage | -plannostage | -planexternal_stage`	Staging mode to use when deploying or distributing a deployment plan independently of an application archive: • `-planstage`—Copies the deployment plan to target servers' staging directories. • `-plannostage`—Does not copy the deployment plan to target servers, but leaves it in a fixed location, specified by the `-plan` option. • `-planexternal_stage`—Does not copy the deployment plan to target servers. Instead, you must ensure that the deployment plan has been copied to the correct subdirectory in the target servers' staging directories. You can manually copy the deployment plan or use a third-party tool or script. If you do not specify a staging mode, the deployment plan uses the value specified for application staging as the default.
`-retiretimeout` *seconds*	Number of seconds before WebLogic Server retires the currently-running version of this application or module. See Redeploying a New Version of an Application.
`-library`	The deployment is a shared Java EE library or optional package. You must include the `-library` option when deploying or distributing any Java EE library or optional package. See Deploying Shared Java EE Libraries and Dependent Applications.
`-libspecver` *version*	Specification version of a Java EE library or optional package. This option can be used only if the library or package does not include a specification version in its manifest file. `-libversion` can be used only in combination with `-library`. See Registering Libraries with WebLogic Server.
`-libimplver` *version*	Implementation version of a Java EE library or optional package. This option can be used only if the library or package does not include an implementation version in its manifest file. `-libimplversion` can be used only in combination with `-library`. See Registering Libraries with WebLogic Server.
`-usenonexclusivelock`	Deployment operation uses an existing lock, already acquired by the same user, on the domain. This option is helpful in environments where multiple deployment tools are used simultaneously and one of the tools has already acquired a lock on the domain configuration.
`-altappdd` *file*	(Deprecated.) Name of an alternate Java EE deployment descriptor (`application.xml`) to use for deployment.

Command or Option	Definition
`-altwlsappdd` *file*	(Deprecated.) Name of an alternate WebLogic Server deployment descriptor (`weblogic-application.xml`) to use for deployment.
`-securityModel [ DDOnly | CustomRoles | CustomRolesAndPolicies | Advanced ]`	Security model to use for this deployment.
`-enableSecurityValidation`	Enable validation of security data.
`-id` *task_id*	Task identifier of a running deployment task. You can specify an identifier with the **`distribute`**, **`deploy`**, **`redeploy`**, **`start`**, or **`undeploy`** commands, and use it later as an argument to the **`cancel`** or **`list`** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.
`-extendLoader`	Distributes a code source such as a *custom.jar* file to the WebLogic Extension Loader, which is also referred to as the WebLogic Domain Loader. The code source file is distributed to all targets and when the operation is complete, classes and resources in the jar file will be visible for class loading at the extension loader. This is an asynchronous operation that returns immediately.
`-dbClientData`	The deployment contains a DBClientData module. You must include the `-dbClientData` option when deploying or distributing a DBClientData module, which can be an archive file or exploded archive directories, that may contain a `tnsnames.ora` file, server trust keystores, client identity keystores, Oracle wallets, and other data that are needed to connect to a database. All file types are optional. For more information, see Use DBClientData Modules for Portability in *Administering JDBC Data Sources for Oracle WebLogic Server*.
`-dbClientDataUploadPath`	Use the `-dbClientDataUploadPath` option to specify a non-default, upload directory, relative to the domain configuration directory, to which to upload and deploy a DBClientData module. The domain configuration directory is `DOMAIN_HOME/config` and for DBClientData modules, the default value for this is `DOMAIN_HOME/config/dbclientdata`.

Examples

See the following sections for examples of using the **`-deploy`** command:

- Deploying to a Single-Server Domain
- Deploying an Application with a Deployment Plan
- Uploading Deployment Files from a Remote Client
- Deploying to One or More Targets
- Deploying to a Cluster Target
- Using Module-Level Targeting for Deploying an Enterprise Application
- Targeting Application-Scoped JMS, JDBC, and WLDF Modules
- Using Sub-Module Targeting with JMS Application Modules
- Using Nostage Mode Deployment
- Using Stage Mode Deployment
- Using External_stage Mode Deployment
- Distributing Applications to a Production Environment

- Registering Libraries with WebLogic Server
- Deploying Applications That Reference Libraries
- Deploying Oracle Wallet Files for JDBC Modules

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. You can start the application in administration mode, or make it available to administration and client requests. While in administration mode, the application can be accessed only by internal clients through a configured administration port. External clients cannot access the application.

- Syntax
- Examples

Syntax

```
java [SSL Arguments] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -distribute [[-name] deployment_name] [-source] file
     [-editsession edit_session_name]
     [-plan file] [-targets target_list] [-submoduletargets target_list]
     [-upload]
     [-stage | -nostage | -external_stage]
     [-planstage | -plannostage | -planexternal_stage]
     [-library [-libspecver version] [-libimplver version]]
     [-altappdd file] [-altwlsappdd file]
     [-securityModel] [-enableSecurityValidation]
     [-id task_id]
     [-dbClientData] [-dbClientDataUploadPath]
     [Common Arguments]
```

Command or Option	Definition
`-name deployment_name`	Deployment name to assign to the distributed application or module. Both the `-name` option and *`deployment_name`* argument are optional, as described in the Syntax. If a deployment name is not explicitly identified, a name is derived from the specified deployment file or directory: • For an archive file, the default deployment name is the name of the archive file without the file extension (`myear` for the file `myear.ear`). • For an exploded archive directory, the default deployment name is the name of the top-level directory. • If you specify an application installation root directory, the default deployment name is derived from the archive file name or exploded archive directory name in the `/app` subdirectory.
`-source file`	Archive file or exploded archive directory to distribute. You can omit the `-source` option and supply only the file or directory.
`-editsession edit_session_name`	Specifies the name of the edit session that was active during this deployment operation. If an edit session is specified, then the deployment will be associated with the edit session and deferred until the edit session is activated.
`-plan file`	Deployment plan to distribute with the application or module, used to configure the application.

Command or Option	Definition
`-targets target_list`	Targets on which to distribute the application or module. The `target_list` argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a Java EE module name (`<module1>@<server1>`). This enables you to distribute different modules of an enterprise application to different servers or clusters. See Using Module-Level Targeting for Deploying an Enterprise Application. If you do not specify a target list with the **distribute** command, the target defaults to: • The Administration Server instance for new deployments. • The application's current targets for deployed applications.
`-submoduletargets target_list`	JMS Server targets for resources defined within a JMS application module. See Using Sub-Module Targeting with JMS Application Modules and Using WLST to Manage JMS Servers and JMS System Resources in *Administering JMS Resources for Oracle WebLogic Server*.
`-upload`	Transfers the specified deployment files, including any specified deployment plans, to the Administration Server before distribution. Use this option when you are on a remote machine and you cannot copy the deployment files to the Administration Server by other means. The application files are uploaded to the WebLogic Server Administration Server upload directory prior to distribution.
`-stage | -nostage | -external_stage`	Staging mode to use when deploying or distributing an application: • `-stage`—Copies deployment files to target servers' staging directories. stage is the default mode used when deploying or distributing to Managed Server targets. • `-nostage`—Does not copy the deployment files to target servers, but leaves them in a fixed location, specified by the `-source` option. Target servers access the same copy of the deployment files. `nostage` is the default used when deploying or distributing to the Administration Server (for example, in a single-server domain). • `-external_stage`—Does not copy the deployment files to target servers; instead, you must ensure that deployment files have been copied to the correct subdirectory in the target servers' staging directories. You can manually copy the files or use a third-party tool or script. See Controlling Deployment File Copying with Staging Modes.
`-planstage | -plannostage | -planexternal_stage`	Staging mode to use when deploying or distributing a deployment plan independently of an application archive: • `-planstage`—Copies the deployment plan to target servers' staging directories. • `-plannostage`—Does not copy the deployment plan to target servers, but leaves it in a fixed location, specified by the `-plan` option. • `-planexternal_stage`—Does not copy the deployment plan to target servers. Instead, you must ensure that the deployment plan has been copied to the correct subdirectory in the target servers' staging directories. You can manually copy the deployment plan or use a third-party tool or script. If you do not specify a staging mode, the deployment plan uses the value specified for application staging as the default.

Command or Option	Definition
`-library`	Identifies the deployment as a shared Java EE library or optional package. You must include the `-library` option when deploying or distributing any Java EE library or optional package. See Deploying Shared Java EE Libraries and Dependent Applications.
`-libspecver` *version*	Specification version of a Java EE library or optional package. This option can be used only if the library or package does not include a specification version in its manifest file. `-libversion` can be used only in combination with `-library`. See Registering Libraries with WebLogic Server.
`-libimplver` *version*	Implementation version of a Java EE library or optional package. This option can be used only if the library or package does not include a implementation version in its manifest file. `-libimplversion` can be used only in combination with `-library`. See Registering Libraries with WebLogic Server.
`-altappdd` *file*	(Deprecated.) Name of an alternate Java EE deployment descriptor (`application.xml`) to use for deployment or distribution.
`-altwlsappdd` *file*	(Deprecated.) Name of an alternate WebLogic Server deployment descriptor (`weblogic-application.xml`) to use for deployment or distribution.
`-securityModel [ DDOnly | CustomRoles | CustomRolesAndPolicies | Advanced ]`	Security model to be used for this application.
`-dbClientData`	The deployment contains a DBClientData module. You must include the `-dbClientData` option when deploying or distributing a DBClientData module, which can be an archive file or exploded archive directories, that may contain a `tnsnames.ora` file, server trust keystores, client identity keystores, Oracle wallets, and other data that are needed to connect to a database. All file types are optional. For more information, see Use DBClientData Modules for Portability in *Administering JDBC Data Sources for Oracle WebLogic Server*.
`-dbClientDataUploadPath`	Use the `-dbClientDataUploadPath` option to specify a non-default, upload directory, relative to the domain configuration directory, from which to distribute a DBClientData module. The domain configuration directory is `DOMAIN_HOME/config` and for DBClientData modules, the default value for this is `DOMAIN_HOME/config/dbclientdata`.

Examples

The **`distribute`** command operates similar to **`deploy`**, but WebLogic Server does not start the application or module on target servers. See the examples links for Deploy command for more information.

Listapps

Lists the deployment names for applications and standalone modules deployed, distributed, or installed to the domain.

- Syntax
- Examples

Syntax

```
java [SSL Arguments] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -listapps
     [Common Arguments]
```

Command or Option	Definition
-listapps	Lists the deployment names for applications and standalone modules deployed, distributed, or installed to the domain

Examples

See Displaying Version Information for Deployed Applications.

List, Listtask

Displays the status of deployment tasks currently running in the domain.

- Syntax
- Examples

Syntax

```
java [SSL Arguments] weblogic.Deployer Connection Arguments
     [User Credentials Arguments] <-list | -listtask> [task_id]
     [Common Arguments]
```

Command or Option	Definition
task_id	Identifier of a deployment task to display.

Examples

See Managing Long-Running Deployment Tasks.

Purgetasks

Indicates that weblogic.Deployer should flush out deployment tasks that are retired.

- Syntax
- Examples

Syntax

```
java [SSL Arguments] weblogic.Deployer Connection Arguments [User Credentials Arguments] -
purgetasks
```

Examples

The following command indicates that weblogic.Deployer should flush out any deployment tasks that are retired:

```
java weblogic.Deployer -username
     weblogic -password ... -adminurl t3://localhost:7001 -purgetasks
:
:
weblogic.Deployer invoked with options:-username weblogic -adminurl
t3://localhost:7001 -purgetasks
Currently there are no retired tasks.
```

Redeploy

Redeploys a running application or part of a running application.

- Syntax
- Examples

Syntax

```
java [SSL Arguments] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -redeploy [[-name] deployment_name] [-plan file]
     [-removePlanOverride]
     [-editsession edit_session_name]
     [-targets target_list] [-submoduletargets target_list]
     [-upload]
     [-delete_files]
     [-retiretimeout seconds] [-id task_id]
     [-rmiGracePeriod seconds]
     [-dbClientData] [-dbClientDataUploadPath]
     [Common Arguments]
```

Command or Option	Definition
`-name deployment_name`	Deployment name of a deployed application or module. The `-name` option can be omitted, in which case the name is taken from the `-source file` argument.
`-source file`	Archive file or exploded archive directory to distribute, deploy, or redeploy. When used with the `redeploy` command, the `-source` option specifies the location of new deployment files to redeploy, for example, when updating an application to a new version or when rolling back to the previously deployed version. **Note**: Do not specify the `redeploy -source` option for non-versioned applications or existing versioned applications if the source location has changed. In those cases, you must first undeploy, then deploy the application. To specify multiple files for a partial redeployment, omit the `-source` option and supply only a *filelist*. **Note**: To redeploy an entire Java EE module within an enterprise application, use the module-targeting syntax, `-targets module@target`, described in Using Partial Redeployment for Java EE Module Updates.
`-removePlanOverride`	Removes an overridden deployment plan during a `redeploy` or `update` deployment action.
`-editsession edit_session_name`	Specifies the name of the edit session that was active during this deployment operation. If an edit session is specified, then the deployment will be associated with the edit session and deferred until the edit session is activated.

Command or Option	Definition
`-plan` *file*	Deployment plan to use when distributing, deploying, or redeploying. When redeploying an application, the `-plan` option allows you to specify an updated configuration to use during the redeployment. If the revised deployment plan contains changes to resource bindings, WebLogic Server attempts to redeploy a new version of the application alongside an older version. See Updating the Deployment Configuration for an Application.
`-targets` *target_list*	Targets on which to redeploy the application or module. The *target_list* argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a Java EE module name (`<module1>@<server1>`). This enables you to redeploy different modules of an enterprise application to different servers or clusters. See Using Module-Level Targeting for Deploying an Enterprise Application. **Note**: To redeploy an entire Java EE module within an enterprise application, use the module-targeting syntax, `-targets` *module@target*, described in Using Partial Redeployment for Java EE Module Updates. If you do not specify a target list with the **redeploy** command, the target defaults to: • The Administration Server instance for new deployments. • The application's current targets for deployed applications. If you do not specify a target list with the **redeploy** command, the application is redeployed on all of its current target servers.
`-submoduletargets` *target_list*	JMS Server targets for resources defined within a JMS application module. See Using Sub-Module Targeting with JMS Application Modules and Using WLST to Manage JMS Servers and JMS System Resources in *Administering JMS Resources for Oracle WebLogic Server*.
`-upload`	Transfers the specified deployment files, including deployment plans and alternate deployment descriptors, to the Administration Server. Use this option when you are on a remote machine and you cannot copy the deployment files to the Administration Server by other means. The application files are uploaded to the WebLogic Server Administration Server upload directory prior to distribution and deployment.
`-delete_files`	Removes static files from a server's staging directory. `delete_files` is valid only for unarchived deployments, and only for applications deployed using `-stage` mode. You must specify target servers when using this option, as shown in the following example: `java weblogic.Deployer -adminurl http://myserver:7001 -username weblogic -password weblogic -name myapp -targets myapp@myserver -redeploy -delete_files myapp/tempindex.html` `delete_files` only removes files that WebLogic Server copied to the staging area during deployment. If you use the `delete_files` option with an application that was deployed using either `-nostage` or `-external_stage` mode, the command does not delete the files. **Note**: `delete_files` can only be used in combination with the **redeploy** command. `Because the -delete_files` option deletes all specified files or, if you specify a directory but do not specify files within the directory, all files in the specified directory, Oracle recommends that you use caution when using the delete_files option and that you do not use the delete_files option in production environments.
`-retiretimeout` *seconds*	Number of seconds before WebLogic Server retires the currently-running version of this application or module. See Redeploying a New Version of an Application.
`-rmiGracePeriod` *seconds*	The amount of time, in seconds, that the work manager accepts and schedules RMI calls until there are no more RMI requests arriving within the RMI grace period during a `graceful` shutdown or a `retirement`.

Command or Option	Definition
`-dbClientData`	The deployment contains a DBClientData module. You must include the `-dbClientData` option when redeploying a DBClientData module, which can be an archive file or exploded archive directories, that may contain a `tnsnames.ora` file, server trust keystores, client identity keystores, Oracle wallets, and other data that are needed to connect to a database. All file types are optional. For more information, see Use DBClientData Modules for Portability in *Administering JDBC Data Sources for Oracle WebLogic Server*.
`-dbClientDataUploadPath`	Use the `-dbClientDataUploadPath` option to specify a non-default, upload directory, relative to the domain configuration directory, from which to redeploy a DBClientData module. The domain configuration directory is `DOMAIN_HOME/config` and for DBClientData modules, the default value for this is `DOMAIN_HOME/config/dbclientdata`.

Examples

See the following sections for examples of using the **`redeploy`** command:

- Redeploying a New Version of an Application
- Rolling Back the Production Redeployment Process
- Steps for Distributing a New Version of an Application
- Redeploying Applications and Modules In-Place
- Using Partial Redeployment for Java EE Module Updates
- Updating Static Files in a Deployed Application
- Redeploy DBClientData Modules

Start

Makes a stopped (inactive) application available to clients on target servers. The **`start`** command does not redistribute deployment files to target servers. Optionally, with the **`adminmode`** option, starts the application in administration mode, which makes it available only via a configured administration channel. In order to issue a `start` command, the files must already be available through a **`deploy`** or **`distribute`** command.

> **Note:**
>
> The **`activate`** command, an alias for **`start`**, is deprecated.

- Syntax
- Examples

Syntax

```
java [SSL Arguments] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -start [-adminmode] [-name] deployment_name
     [-editsession edit_session_name]
     [-appversion version] [-planversion version]
     [-targets target_list] [-submoduletargets target_list]
```

```
[-retiretimeout seconds]
[-id task_id]
[Common Arguments]
```

Command or Option	Definition
`-adminmode`	Start application in administration mode, not Production mode (which is the default).
`-name deployment_name`	Deployment name of a deployed application or module. The `name` option can be omitted, in which case the name is taken directly from the `deployment_name`. (If the `deployment_name` specifies a file or directory name, the deployment name is derived from the file specification.)
`-editsession edit_session_name`	Specifies the name of the edit session that was active during this deployment operation. If an edit session is specified, then the deployment will be associated with the edit session and deferred until the edit session is activated.
`-appversion version`	Version of the application to start.
`-planversion version`	Version of the deployment plan to use when starting the application.
`-targets target_list`	Targets on which to start the application or module. The *target_list* argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a Java EE module name (`<module1>@<server1>`). This enables you to start different modules of an enterprise application on different servers or clusters. See Using Module-Level Targeting for Deploying an Enterprise Application. If you do not specify a target list with the **start** command, the target defaults to: • The Administration Server instance for new deployments. • The application's current targets for deployed applications. If you do not specify a target list with the **start** command, the command is performed on all of the application's current targets.
`-submoduletargets target_list`	JMS Server targets for resources defined within a JMS application module. See Using Sub-Module Targeting with JMS Application Modules and Using WLST to Manage JMS Servers and JMS System Resources in *Administering JMS Resources for Oracle WebLogic Server*.
`-retiretimeout seconds`	Number of seconds before WebLogic Server retires the currently-running version of this application or module. See Redeploying a New Version of an Application.
`-id task_id`	Task identifier of a running deployment task. You can specify an identifier with the **distribute**, **deploy**, **redeploy**, **start**, or **undeploy** commands, and use it later as an argument to the **cancel** or **list** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.

Examples

See the following sections for examples of using the **`start`** command:

- Starting a Distributed Application
- Making an Application Available to Clients
- Stopping an Application to Restrict Client Access

Stop

Makes an application inactive and unavailable administration and client requests. All of the application's staged files remain available on target servers for subsequent **`start`**, **`deploy`**, **`redeploy`**, or **`undeploy`** actions. Optionally, choose to make the application unavailable to client

requests by placing it in administration mode with the **adminmode** option. While in administration mode, the application be accessed only through a configured administration channel.

> **Note:**
>
> The **deactivate** command, an alias for **stop**, is deprecated.

- Syntax
- Examples

Syntax

```
java [SSL Arguments] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -stop [-adminmode] [-name] deployment_name
     [-editsession edit_session_name]
     [-appversion version] [-planversion version]
     [-targets target_list] [-submoduletargets target_list]
     [-ignoresessions] [-graceful] [-rmiGracePeriod seconds]
     [-id task_id]
     [Common Arguments]
```

Argument or Option	Definition
`-adminmode`	Indicates that a running application should switch to administration mode and accept only administration requests via a configured administration channel. If this option is not specified, the running application is stopped and cannot accept administration or client requests until is it restarted.
`-name` *deployment_name*	Deployment name of a deployed application or module. The `name` option can be omitted, in which case the name is taken directly from the `deployment_name`. (If the `deployment_name` specifies a file or directory name, the deployment name is derived from the file specification.)
`-editsession` *edit_session_name*	Specifies the name of the edit session that was active during this deployment operation. If an edit session is specified, then the deployment will be associated with the edit session and deferred until the edit session is activated.
`-appversion` *version*	Version identifier of the deployed application.
`-planversion` *version*	Version identifier of the deployment plan.
`-targets` *target_list*	Targets on which to stop the application or module. The *target_list* argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a Java EE module name (<module1>@<server1>). This enables you to stop different modules of an enterprise application on different servers or clusters. See Using Module-Level Targeting for Deploying an Enterprise Application. If you do not specify a target list with the **stop** command, the target defaults to: • The Administration Server instance for new deployments. • The application's current targets for deployed applications. If you do not specify a target list with the **stop** command, the command is performed on all of the application's current targets.
`-submoduletargets` *target_list*	JMS Server targets for resources defined within a JMS application module. See Using Sub-Module Targeting with JMS Application Modules and Using WLST to Manage JMS Servers and JMS System Resources in *Administering JMS Resources for Oracle WebLogic Server*.

Argument or Option	Definition
`-graceful`	Stops the application after existing HTTP clients have completed their work. If you do not specify the `-graceful` option, WebLogic Server immediately stops the application or module. See Taking a Production Application Offline.
`-rmiGracePeriod` *seconds*	The amount of time, in seconds, that the work manager accepts and schedules RMI calls until there are no more RMI requests arriving within the RMI grace period during a `graceful` shutdown or a `retirement`.
`-ignoresessions`	Immediately places the application into administration mode without waiting for current HTTP sessions to complete.
`-id` *task_id*	Task identifier of a running deployment task. You can specify an identifier with the **distribute**, **deploy**, **redeploy**, **start**, **stop**, or **undeploy** commands, and use it later as an argument to the **cancel** or **list** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.

Examples

See Stopping an Application to Restrict Client Access.

Undeploy

Stops the deployment unit and removes staged files from target servers.

> **Note:**
>
> The **`REMOVE`** command, an alias for **`undeploy`**, is deprecated.

> **Note:**
>
> When you undeploy an application that contains application-scoped resources, the resources are deleted along with the application, which can potentially cause abandoned transactions or lost messages as a result of deleted JMS destinations. See Unregister Resource Grace Period in *Developing JTA Applications for Oracle WebLogic Server*.
>
> You should only undeploy applications that you are certain you want to completely remove; to temporarily stop client access to applications, use the **`stop`** command, described in Stop, instead.

- Syntax
- Examples

Syntax

```
java [SSL Arguments] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -undeploy [-name] deployment_name
```

```
[-editsession edit_session_name]
[-appversion version] [-planversion version]
[-targets target_list] [-submoduletargets target_list]
[-graceful] [-ignoresessions] [-rmiGracePeriod seconds]
[-id task_id]
[Common Arguments]
```

Command or Option	Definition
`-name deployment_name`	Deployment name of a deployed application or module. The `name` option can be omitted, in which case the name is taken directly from the `deployment_name`. (If the `deployment_name` specifies a file or directory name, the deployment name is derived from the file specification.)
`-editsession edit_session_name`	Specifies the name of the edit session that was active during this deployment operation. If an edit session is specified, then the deployment will be associated with the edit session and deferred until the edit session is activated.
`-appversion version`	Version identifier of the deployed application.
`-planversion version`	Version identifier of the deployment plan.
`-targets target_list`	Targets from which the application or module are undeployed. **Note**: Any target not included in the target list is *not* removed. The `target_list` argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a Java EE module name (<module1>@<server1>). This enables you to undeploy different modules of an enterprise application from different servers or clusters. See Using Module-Level Targeting for Deploying an Enterprise Application.
`-submoduletargets target_list`	JMS resources to be undeployed. **Note**: Any sub-module target not included in the target list is *not* removed. See Using Sub-Module Targeting with JMS Application Modules and Using WLST to Manage JMS Servers and JMS System Resources in *Administering JMS Resources for Oracle WebLogic Server*.
`-graceful`	Stops the application after existing HTTP clients have completed their work. If you do not specify the -graceful option, WebLogic Server immediately stops the application or module. See Taking a Production Application Offline. The module is undeployed after it is stopped.
`-rmiGracePeriod seconds`	The amount of time, in seconds, that the work manager accepts and schedules RMI calls until there are no more RMI requests arriving within the RMI grace period during a `graceful` shutdown or a `retirement`.
`-ignoresessions`	Immediately stops and undeploys the application without waiting for current HTTP sessions to complete.
`-id task_id`	Task identifier of a running deployment task. You can specify an identifier with the **distribute**, **deploy**, **redeploy**, **start**, **stop**, or **undeploy** commands, and use it later as an argument to the **cancel** or **list** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.

Examples

See the following sections for examples of using the **`undeploy`** command:

- Undeploying an Application or Module
- Sub-module Targeting for Standalone JMS Modules
- Undeploy DBClientData Modules

Update

Updates an application's deployment plan by redistributing the plan files and reconfiguring the application based on the new plan contents.

> **Note:**
>
> **update** cannot be used to update an application's resource bindings. To update the resource bindings for an application, you must use the Redeploy command.

- Syntax
- Example

Syntax

```
java [SSL Arguments] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -update -plan deployment_plan [-name] deployment_name
     [-removePlanOverride]
     [-editsession edit_session_name]
     [-appversion version] [-planversion version]
     [-targets target_list] [-submoduletargets target_list]
     [-upload] [-id task_id]
     [Common Arguments]
```

Argument or Option	Definition
`-plan deployment_plan`	Deployment plan to use for updating the application's configuration. The specified deployment plan must be valid for the application's target servers. For example, the plan cannot contain null variables for required resources unless those resources were previously defined in the associated descriptors. Update operations update only those descriptors for which there is a changed, not null value in the deployment plan. If a plan that is used by an update operation contains null variables, the current values in the corresponding descriptors are not updated.
`-name deployment_name`	Deployment name of a deployed application or module. The `name` option can be omitted, in which case the name is taken directly from the `deployment_name`. (If the `deployment_name` specifies a file or directory name, the deployment name is derived from the file specification.)
`-removePlanOverride`	Removes an overridden deployment plan during a `redeploy` or `update` deployment action.
`-editsession edit_session_name`	Specifies the name of the edit session that was active during this deployment operation. If an edit session is specified, then the deployment will be associated with the edit session and deferred until the edit session is activated.
`-appversion version`	Version identifier of the deployed application.
`-planversion version`	Version identifier of the deployment plan.

Argument or Option	Definition
`-targets target_list`	Targets on which to update the application or module. The `target_list` argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a Java EE module name (<module1>@<server1>). This enables you to update different modules of an enterprise application on different servers or clusters. See Using Module-Level Targeting for Deploying an Enterprise Application. If you do not specify a target list with the **update** command, the target defaults to: • The Administration Server instance for new deployments. • The application's current targets for deployed applications. If you do not specify a target list with the **update** command, the command is performed on all of the application's current targets.
`-submoduletargets target_list`	JMS Server targets for resources defined within a JMS application module. See Using Sub-Module Targeting with JMS Application Modules and Using WLST to Manage JMS Servers and JMS System Resources in *Administering JMS Resources for Oracle WebLogic Server*.
`-upload`	Uploads a new deployment plan to the Administration Server before updating the application.
`-id task_id`	Task identifier of a running deployment task. You can specify an identifier with the **distribute**, **deploy**, **redeploy**, **update**, **start**, **stop**, or **undeploy** commands, and use it later as an argument to the **cancel** or **list** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.

Example

See Updating an Application to Use a Different Deployment Plan.

Exit Codes

An exit code of zero (0) indicates success, whereas, a non-zero exit code indicates one or more failures were detected, the value being the number of failures.

In general, an exit code of 1 is returned if the command was rejected. Rejection is due to syntax errors, an invalid ID option, connection errors (for example, no Administration Server response), source errors (for example, file not found), or undefined targets.

Example config.xml File and Corresponding weblogic.Deployer Command

Examine an application's `config.xml` file and the corresponding `weblogic.Deployer` command to deploy the application.

Assuming:

- `mycluster` is a cluster name
- `D1C2S1` and `D1C2S2` are server names
- `RemoteJMSServer1` and `RemoteJMSServer2` are JMS server names

The application's `config.xml` file would contain:

```
<app-deployment>
   <name>dd-remote-cluster</name>
   <source-path>./udd-debug-deployment-on-remote-cluster-jms.xml</source-path>
   <target>mycluster</target>
   <subdeployment>
      <name>RemoteCluster</name>
      <target>mycluster</target>
   </subdeployment>
   <subdeployment>
      <name>D1C2S2</name>
      <target>D1C2S2</target>
   </subdeployment>
   <subdeployment>
      <name>RemoteClusterServers</name>
      <target>D1C2S1,D1C2S2</target>
   </subdeployment>
   <subdeployment>
      <name>RemoteClusterJMSServers</name>
      <target>RemoteJMSServer1,RemoteJMSServer2</target>
   <subdeployment>
      <name>RemoteQueue1</name>
      <target>RemoteJMSServer1</target>
</app-deployment>
```

The `weblogic.Deployer -deploy` command to deploy the application would be:

```
  java weblogic.Deployer -adminurl t3://MySystem:10000 -username system
-password password -name dd-remote-cluster
-deploy "config\jms\udd-debug-deployment-on-remote-cluster-jms.xml"
-targets mycluster -submoduletargets RemoteCluster@mycluster,
D1C2S2@D1C2S2, RemoteClusterServers@D1C2S1,
RemoteClusterServers@D1C2S2, RemoteClusterJMSServers@RemoteJMSServer1,
RemoteClusterJMSServers@RemoteJMSServer2, RemoteQueue1@RemoteJMSServer1
```

B

weblogic.PlanGenerator Command-Line Reference

Learn how to use the `weblogic.PlanGenerator` utility, a Java-based deployment configuration tool intended for developers who want to export portions of a WebLogic Server deployment configuration into a deployment plan.

> **Note:**
>
> See also *Understanding the WebLogic Scripting Tool* for information about performing deployment configuration operations using the WebLogic Scripting Tool (WLST).

This appendix includes the following sections:

- Overview of weblogic.PlanGenerator
  The `weblogic.PlanGenerator` utility generates WebLogic Server deployment configuration files for an application or standalone module.
- Required Environment for weblogic.PlanGenerator
  Set up your environment to use the `weblogic.PlanGenerator` utility.
- Syntax for Invoking weblogic.PlanGenerator
- Common weblogic.PlanGenerator Tasks
  Examine common configuration and export tasks.

Overview of weblogic.PlanGenerator

The `weblogic.PlanGenerator` utility generates WebLogic Server deployment configuration files for an application or standalone module.

`weblogic.PlanGenerator` provides two primary functions:

- Exporting different categories of WebLogic Server deployment descriptor properties into empty (null) variables in a template deployment plan. You can optionally use an existing deployment plan as input to the configuration export session. Template plans are generally modified further before they can be used. See Exporting an Application for Deployment to New Environments.
- Generating a simple initial deployment plan from a Jakarta EE application. See Generating a Template Deployment Plan using weblogic.PlanGenerator.

By default, `weblogic.PlanGenerator` writes an application's deployment plan to a file named `plan.xml` in the application's root directory. If your application is not in an application root directory, `weblogic.PlanGenerator` writes `plan.xml` to `<your_dir>/config/deployments/<user>/<application_name>/plan` where:

- *your_dir* is your TEMP directory if it is specified in the property java.io.tmpdir. If the property java.io.tmpdir is not specified, *your_dir* is the WebLogic Server domain directory.
- *user* is your user name.
- *application* is the name of the application.

Required Environment for weblogic.PlanGenerator

Set up your environment to use the weblogic.PlanGenerator utility.

1. Install and configure the WebLogic Server software, as described in the *Installing and Configuring Oracle WebLogic Server and Coherence*.
2. Add the WebLogic Server classes to the CLASSPATH environment variable, and ensure that the correct JDK binaries are available in your PATH. 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 environment.

> **Note:**
>
> On UNIX operating systems, the setWLSEnv.sh command does not set the environment variables in all command shells. Oracle recommends that you execute this command using the Korn shell or bash shell.

Syntax for Invoking weblogic.PlanGenerator

```
java weblogic.PlanGenerator [Options] [filespec]
```

The *filespec* can be either:

- An absolute or relative path to an archive file
- An absolute or relative path to an exploded archive directory

Oracle recommends:

- That you store your applications in an installation root directory.
- That you specify an application's installation root directory, with the -root option as described in Options when you issue a weblogic.PlanGenerator command.

In all cases, the application identified with the *filespec* must contain valid Jakarta EE deployment descriptor files.

If you do not specify an application root directory with the -root option or a deployment plan path and name with the -plan option, by default, weblogic.PlanGenerator writes an application's deployment plan to a file named plan.xml in the application's root directory. If it cannot locate an application root directory, weblogic.PlanGenerator writes plan.xml to <*your_dir*>/config/deployments/<*user*>/<*application_name*>/plan where:

- *your_dir* is your TEMP directory if it is specified in the property java.io.tmpdir. If the property java.io.tmpdir is not specified, *your_dir* is the WebLogic Server domain directory.
- *user* is your user name.

- *application* is the name of the application.
- Options

Options

The following table describes each `weblogic.PlanGenerator` option.

Table B-1 weblogic.PlanGenerator Options

Option	Description
`-debug`	Enables debug mode.
`-plan` *plan_file*	Identifies the path and name of the plan file to create for the configuration session.
`-useplan` *plan_file*	Existing deployment plan file to initialize from. If you use `-root` to specify an application root directory, `weblogic.PlanGenerator` uses the `/plan/plan.xml` file in the root directory as input, if one is available; otherwise, `weblogic.PlanGenerator` creates a plan in the `root` directory. weblogic.PlanGenerator adds additional exported properties to the input plan; any exported properties that were already in the input plan are retained.
`-root` *root_directory*	Application root directory on which to perform the plan generation or export.
`category` where valid values for `category` are: • `-all` • `-any` • `-configurables` • `-dependencies` • `-declarations` • `-dynamics` • `-none` • `-standard`	Generates null variable definitions in a template deployment plan for different categories of deployment configuration property: • `all`—Creates a plan that exports all editable properties. • `any`—Creates a plan that exports all properties. • `configurables`—Creates a plan that exports all editable properties except dependencies and declarations. • `dependencies`—Creates a plan that exports all WebLogic Server descriptor properties that resolve external resource references. This is the default value. • `declarations`—Creates a plan that exports all properties that declare a resource to other applications and modules. • `dynamics`—Creates a plan that exports all properties that can be changed on-the-fly without requiring the application to be redeployed. • `none`—Creates a plan that exports no properties. • `standard`—Creates a plan that includes standard descriptors.
`-variables [global | unique]`	Specifies whether variable names created in the deployment plan can be used across all modules of an application, or only within a particular module. For example, a role assignment might be applied to an entire EAR, or to only a single Web application or other module within the EAR. By default, `PlanGenerator` creates global variables that apply to the entire application.
`-noexit`	Exceptions occur instead of exiting.

Table B-1 (Cont.) weblogic.PlanGenerator Options

Option	Description
`-module module_name`	Creates a plan only for a module.
`-library </mylibs/lib.ear@name=mylib, /mylibs/lib2.ear@name= otherlib@libspecver=1@libimplver=2>`	Comma-separated specifications of libraries to merge.
`-librarydir library_directory`	Location of directory of libraries to merge.

Common weblogic.PlanGenerator Tasks

Examine common configuration and export tasks.

The following sections provide examples using `weblogic.PlanGenerator` syntax.

- Creating an Initial Deployment Plan in an Application's Root Directory
- Creating a New Deployment Plan Based on an Existing Plan
- Controlling the Components Exported to a Deployment Plan

Creating an Initial Deployment Plan in an Application's Root Directory

If you store your application using an installation root directory, and you specify the root directory with the `-root` option, generated deployment plan files are automatically stored in the root directory's `plan` subdirectory.

```
java weblogic.PlanGenerator -root /appRelease/MyApplication
```

In the above example, the `plan.xml` file is automatically stored in `/appRelease/MyApplication/plan`.

Creating a New Deployment Plan Based on an Existing Plan

The following command uses an existing plan as input and generates a new plan in the `/plan` subdirectory of the application root directory:

```
java weblogic.PlanGenerator -useplan /plans/MyApplication_template.xml
     -root /appRelease/MyApplication
```

Controlling the Components Exported to a Deployment Plan

You can use the `-all`, `-configurables`, `-dependencies`, `-declarations`, `-dynamics`, and `-none` options to specify the WebLogic Server deployment descriptor components that are exported to a template deployment plan. The following command exports all configurable properties to null variables in a template deployment plan:

```
java weblogic.PlanGenerator -root /appRelease/MyApplication -all
```

See Exporting an Application for Deployment to New Environments for more information about exporting a deployment configuration.