# Oracle® Fusion Middleware

## Understanding Oracle WebLogic Server

15c (15.1.1.0.0)
G31428-01
October 2025

Oracle Fusion Middleware Understanding Oracle WebLogic Server, 15c (15.1.1.0.0)

G31428-01

# Contents

# Preface

This document provides an overview of Oracle WebLogic Server 15c features and describes how you can use them to create enterprise-ready solutions.

## Audience

This document is intended for anyone interested in an overview of the key concepts and architecture of Oracle WebLogic Server.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

**Access to Oracle Support**

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://support.oracle.com/portal/ or visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab if you are hearing impaired.

## 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.

## 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

# Introduction

Oracle WebLogic Server is the industry's best application server for building and deploying Jakarta EE applications with support for new features for lowering cost of operations, improving performance, enhancing scalability, and supporting the Oracle Applications portfolio.

The following sections provide an overview of Oracle WebLogic Server features and describe how you can use them to create enterprise-ready solutions:

## Product Overview

Oracle WebLogic Server provides a modern development platform for building applications, a runtime platform for high performance and availability, and rich management tooling for efficient and low cost operations.

The WebLogic Server complete implementation of the Jakarta Platform, Enterprise Edition 9.1 (Jakarta EE 9.1) specification provides a standard set of APIs for creating distributed Jakarta applications that can access a wide variety of services, such as databases, messaging services, and connections to external enterprise systems. End-user clients access these applications using web browser clients or Jakarta clients. See Programming Models.

The WebLogic Server infrastructure supports the deployment of many types of distributed applications and is an ideal foundation for building applications based on Service Oriented Architectures (SOA). SOA is a design methodology aimed at maximizing the reuse of application services. See Oracle SOA - Service-Oriented Architecture.

In addition to the Jakarta EE implementation, WebLogic Server enables enterprises to deploy mission-critical applications in a robust, secure, highly available, and scalable environment. These features allow enterprises to configure clusters of WebLogic Server instances to distribute load, and provide extra capacity in case of hardware or other failures. New diagnostic tools allow system administrators to monitor and tune the performance of deployed applications and the WebLogic Server environment itself. You can also configure WebLogic Server to monitor and tune application throughput automatically without human intervention. Extensive security features protect access to services, keep enterprise data secure, and prevent malicious attacks.

## Programming Models

Oracle WebLogic Server provides complete support for the Jakarta Platform, Enterprise Edition 9.1 (Jakarta EE 9.1) Platform, which reduces the complexity of enterprise application development by providing a development model, API, and runtime environment that allow developers to concentrate on functionality. The Jakarta EE 9.1 specification is available at https://jakarta.ee/specifications/platform/9.1/. For information about Jakarta EE programming model support in WebLogic Server, see the following programming guides:

- Web Applications provide the basic Jakarta EE mechanism for deployment of dynamic web pages based on the Jakarta EE standards of servlets and Jakarta Server Pages (JSP). Web applications are also used to serve static web content such as HTML pages and image files.

- Web Services provide a shared set of functions that are available to other systems on a network and can be used as a component of distributed web-based applications.
- XML capabilities include data exchange, and a means to store content independent of its presentation, and more.
- Jakarta Messaging (JMS) enables applications to communicate with one another through the exchange of messages. A message is a request, report, and/or event that contains information needed to coordinate communication between different applications.
- JDBC provides pooled access to DBMS resources.
- Resource Adapters provide connectivity to Enterprise Information Systems (EISes).
- Jakarta Enterprise Beans (EJB) provide Java objects to encapsulate data and business logic.
- Remote Method Invocation (RMI) is the Jakarta standard for distributed object computing, allowing applications to invoke methods on a remote object locally.
- Using the Jakarta EE Security API allow you to integrate authentication and authorization into your Jakarta EE applications. You can also use the Security Provider APIs to create your own custom security providers.
- WebLogic Tuxedo Connectivity (WTC) provides interoperability between WebLogic Server applications and Tuxedo services. WTC allows WebLogic Server clients to invoke Tuxedo services and Tuxedo clients to invoke EJBs in response to a service request.
- Coherence provides distributed caching and data grid capabilities for WebLogic Server applications.
- Overview of WebLogic Server Application Development describes developer tools and best practices for coding WebLogic Server applications.

# High Availability

Oracle WebLogic Server provides several features and tools to support the deployment of highly-available applications that can be automatically scaled to meet demand in a reliable and fault-tolerant manner. The high availability features provided in WebLogic Server include the following:

- WebLogic Server clusters provide scalability and reliability for your applications by distributing the work load among multiple instances of WebLogic Server. Incoming requests can be routed to a WebLogic Server instance in the cluster based on the volume of work being processed. In case of hardware or other failures, session state is available to other cluster nodes that can resume the work of the failed node. In addition, you can implement clusters so that services may be hosted on a single machine with options to migrate the service to another node in the event of failure.

  In addition to replicating HTTP session state across servers within a cluster, WebLogic Server can also replicate HTTP session state across multiple clusters, thereby expanding availability and fault tolerance in multiple geographic regions, power grids, and internet service providers.
- Elasticity in dynamic clusters enables the automatic scaling of dynamic clusters and re-provisioning of their associated resources. The elasticity framework leverages the WebLogic Diagnostic Framework (WLDF) policies and actions system.
- Coherence clusters provide scalability and fault tolerance by distributing data across any number of cluster members ensuring that data is always available and easily accessed by any application hosted in WebLogic Server.

In addition, web applications can choose to use a Coherence data grid for storing and replicating HTTP session state to improve scalability, fault tolerance, and performance.

- Work Managers prioritize work based on rules you define and by monitoring actual run time performance statistics. This information is then used to optimize the performance of your application. Work Managers may be applied globally to a WebLogic Server domain or to a specific application or component.
- Overload protection gives WebLogic Server the ability to detect, avoid, and recover from overload conditions.
- Network channels facilitate the effective use of network resources by segregating network traffic into channels based on the type of traffic.
- Simplified JMS cluster configuration and high availability allows applications to easily scale WebLogic JMS services such as JMS servers, SAF agents, and persistent stores. Cluster-targeted JMS servers and persistent stores allow targeting the JMS service artifacts directly to the cluster and eliminate the need to configure artifacts individually for every server in a cluster.
- WebLogic Server persistent store is a built-in, high-performance storage solution for WebLogic Server subsystems and services that require persistence. For example, it can store persistent JMS messages or temporarily store messages sent using the Store-and-Forward feature. The persistent store supports persistence to a file-based store or to a JDBC-enabled database.
- Store-and-forward services enable WebLogic Server to deliver messages reliably between applications that are distributed across WebLogic Server instances. If the message destination is not available at the moment the messages are sent, either because of network problems or system failures, then the messages are saved on a local server instance and are forwarded to the remote destination once it becomes available.
- Enterprise-ready deployment tools facilitate deployment and migration of applications from the development phase to a production environment.
- Production redeployment enables enterprises to deploy a new version of their application without interrupting work in progress on the older version.

# Diagnostic Framework

The WebLogic Diagnostic Framework is a monitoring and diagnostic service that lets you create, collect, analyze, archive, and access diagnostic data generated by a running WebLogic Server instance and applications deployed on it. This diagnostic data provides insight into the runtime performance of WebLogic Server instances, and deployed applications, which helps you isolate and diagnose faults and performance bottlenecks when they occur. See What Is the WebLogic Diagnostics Framework? in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

# Security

The WebLogic Server security architecture provides a comprehensive, flexible security infrastructure designed to address the security challenges of making applications available on the Internet. WebLogic security can be used standalone to secure WebLogic Server applications, or as part of an enterprise-wide, security management system that represents a best-in-breed security management solution. See Overview of the WebLogic Security Service in *Understanding Security for Oracle WebLogic Server* for more information.

# Client Options

In addition to support for browser-based web application clients, WebLogic Server also supports a variety of client types for creating rich GUI applications or simple command-line utilities. These client types include T3 clients, Java IIOP clients, CORBA clients, JMX clients, JMS clients, Web Services clients, and WebLogic Tuxedo Connector clients. For more information, see Overview of Standalone Clients in *Developing Standalone Clients for Oracle WebLogic Server*.

# Integration with Other Systems

WebLogic Server provides a variety of tools to integrate your applications with disparate systems. These tools include web services, resource adapters, the JMS .NET client, the JMS C client, tooling for integrating JMS providers options, advanced queuing, and RMI.

# Integration with Web Servers

WebLogic Server can be used with web server plug-ins, provided separately, that allow requests to Oracle WebLogic Server to be proxied from Oracle HTTP Server, Apache HTTP Server or Microsoft Internet Information Server. Typically, these web servers serve static HTML content, while requests for dynamic web content, such as JSPs, are directed to the WebLogic Server environment.

# Running Oracle WebLogic Server in Kubernetes

WebLogic Server running in Kubernetes lets you modernize your deployments and reap the benefits of running in Kubernetes.

Some of these benefits include leveraging a modern infrastructure, a reduced time for deployment, automated and simplified patching, automated and accelerated application updates, built-in high availability, dynamic and automated scaling, and portability across clouds and environments.

The WebLogic Kubernetes Toolkit lets you migrate your existing applications, manage and update your domains, deploy and update your applications, monitor them, persist the logs, and automate the creation and patching of images. Integration between the tools lets you automate updates through CI/CD processes. For more information, see the Oracle WebLogic Kubernetes Toolkit.

The following document provides Oracle recommended procedures to obtain, create, and update WebLogic Server and Oracle Fusion Middleware (FMW) images with patches, and to update existing containers running in production. See Obtaining, Creating, and Updating Oracle Fusion Middleware Images with Patches in *Patching with OPatch*.

# WebLogic Deploy Tooling

WebLogic Deploy Tooling (WDT) simplifies the automation of WebLogic Server domain provisioning and applications deployment.

WDT creates a declarative, metadata model that describes the domain, applications, and resources used by applications. This metadata model makes it easy to provision, deploy, and perform domain lifecycle operations in a repeatable fashion. You can use WDT to migrate on-premises domain configuration and applications to a Docker image or a persistent volume in

Kubernetes. Find the complete WebLogic Deploy Tooling documentation and samples, and the open source WebLogic Deploy Tooling GitHub project at https://github.com/oracle/weblogic-deploy-tooling.

# WebLogic Image Tool

The WebLogic Image Tool lets you automate building, patching, and updating your WebLogic Server Docker images, including your own customized images.

With the WebLogic Image Tool, you can:

- Create a customized WebLogic Server and FMW Infrastructure Docker image.
- Patch a base install image of WebLogic Server or FMW Infrastructure.
- Create an auxiliary image containing the WDT model, WDT variables, and WDT archive files.

In addition, you can incorporate these use cases into an automated process for patching and updating your WebLogic Server infrastructure and applications running in Kubernetes. Find the complete WebLogic Image Tool documentation and samples, and the open source WebLogic Image Tool GitHub project at https://github.com/oracle/weblogic-image-tool.

# WebLogic Kubernetes Operator

The WebLogic Kubernetes Operator is an application-specific controller that extends Kubernetes to create, configure, and manage instances of complex applications. The operator follows the standard Kubernetes operator pattern, and simplifies the management and operation of WebLogic domains and deployments.

The operator uses a common set of Kubernetes APIs to provide an improved user experience when automating operations such as provisioning, life cycle management, application versioning, product patching, scaling, and security.

The operator is developed as an open source project fully supported by Oracle. The fastest way to experience the operator is to follow the Quick Start guide. Alternatively, you can peruse the documentation, read the blogs, or try out the samples. For project scripts, additional samples, and source files, see the WebLogic Kubernetes Operator GitHub repository.

# WebLogic Remote Console

The Oracle WebLogic Remote Console is a lightweight, open source console that you can use to manage domain configurations of WebLogic Server Administration Servers or WebLogic Deploy Tooling (WDT) metadata models.

The WebLogic Remote Console provides a WebLogic Server administration GUI that enables REST-based access to WebLogic management information, in alignment with current cloud-native trends.

For more information, see the *Oracle WebLogic Remote Console Online Help*.

> **Note**
>
> The Oracle WebLogic Server Administration Console, a GUI for managing WebLogic domains, was removed in 14.1.2.0.0. The WebLogic Remote Console provides comparable functionality to the removed Administration Console.

# WebLogic Kubernetes Toolkit UI

The WebLogic Kubernetes Toolkit (WKT) UI provides a graphical user interface that wraps the WKT tools, Helm, and the Kubernetes client to simplify the combined use of these tools for Kubernetes deployments.

For more information, see the WebLogic Kubernetes Toolkit UI Documentation in GitHub.

# WebLogic Monitoring Exporter

The WebLogic Monitoring Exporter is a web application that you can deploy on a WebLogic Server instance that you want to monitor. The exporter uses the WebLogic Server RESTful Management Interface for accessing runtime state and metrics and then exports Prometheus-compatible metrics, which can be displayed in Grafana dashboards for monitoring.

For practical examples, see these blog posts: Exporting Metrics from WebLogic Server and Using Prometheus and Grafana to Monitor WebLogic Server on Kubernetes. For a detailed description of the WebLogic Monitoring Exporter, see the WebLogic Monitoring Exporter project in GitHub.

# WebLogic Server API Examples and Sample Applications

Oracle WebLogic Server includes a comprehensive set of code examples and sample applications that show how to implement Jakarta EE APIs and Oracle WebLogic Server-specific features in your applications.
Code examples demonstrating Jakarta EE APIs and other WebLogic Server features are provided in a separate, examples JAR file, which you install in the same *ORACLE_HOME* as your WebLogic Server installation. To access the code examples, launch the `startWebLogic.cmd` or `startWebLogic.sh` command from *ORACLE_HOME*`/user_projects/domains/wl_server`, where *ORACLE_HOME* is the directory you specified as the Oracle Home when you installed Oracle WebLogic Server. As they become available, you can also download additional examples. For more information, see Installing the WebLogic Server Code Examples.

Along with the code examples, two versions of a complete sample application, called Avitek Medical Records (or MedRec), are also available.

The original MedRec is a WebLogic Server sample application suite that concisely demonstrates all aspects of the Jakarta EE platform. MedRec is designed as an educational tool for all levels of Jakarta EE developers. It showcases the use of each Jakarta EE component and illustrates best practice design patterns for component interaction and client development. MedRec also illustrates best practices for developing applications on WebLogic Server.

The Spring version of MedRec, called MedRec-Spring is MedRec recast using the Spring Framework. If you are developing Spring applications on WebLogic Server, you should review the MedRec-Spring sample application. In order to illustrate how Spring can take advantage of the enterprise features of WebLogic Server, MedRec was rearchitected to replace core Jakarta EE components with their Spring counterparts. The functionality in the original version of MedRec is reimplemented using Spring in MedRec-Spring. Refer to the MedRec-Spring sample for details.

To launch MedRec, run `startWebLogic.cmd` or `startWebLogic.sh` command from *ORACLE_HOME*`/user_projects/domains/`*medrec*, where *ORACLE_HOME* is the directory you specified as the Oracle Home when you installed Oracle WebLogic Server.

To launch MedRec-Spring, run the `startWebLogic.cmd` or `startWebLogic.sh` script from *`ORACLE_HOME`*`/user_projects/domains/medrec-spring`, where *`ORACLE_HOME`* is the directory you specified as the Oracle Home when you installed Oracle WebLogic Server.

For detailed instructions, see Installing the WebLogic Server Code Examples.

# Upgrade

WebLogic Server provides robust upgrade capabilities to support migrating your application environment from one version of WebLogic Server to the next. An application environment includes a WebLogic domain and any applications and application data associated with the domain. It may also include external resources, such as firewalls, load balancers, and LDAP servers. Tools and documentation are provided to help you migrate applications implemented on earlier versions of WebLogic Server to the current WebLogic Server environment. See *Upgrading Oracle WebLogic Server*.

2

# System Administration

System administration of WebLogic Server includes a wide range of tasks: creating WebLogic Server domains, deploying applications, migrating domains from development environments to production environments, monitoring and managing the performance of the runtime system, configuring and managing security for applications and system resources, diagnosing and troubleshooting problems, and more.

This chapter includes the following topics:

## Overview of WebLogic Server System Administration

WebLogic Server provides several tools for system administrators: the browser-based WebLogic Remote Console, the WebLogic Scripting Tool (WLST), a scripting language for automation of WebLogic system administration tasks based on Jython; a robust set of RESTful management interfaces; SNMP; the Configuration Wizard; and several command-line utilities.

Because the WebLogic Server management system is based on Jakarta EE and other standards, it integrates with systems that are frequently used to manage other software and hardware components. In addition, WebLogic Server implements the JMX specification, which allows programmatic access to the WebLogic Server management system. Using this API, you can create custom administration utilities or automate frequent tasks using Java classes.

The following sections provide an overview of system administration for the WebLogic Server component of your development or production environments:

- Choosing the Appropriate Technology for Your Administrative Tasks
- Summary of System Administration Tools and APIs

For information about installing WebLogic Server, see *Installing and Configuring Oracle WebLogic Server and Coherence*.

## Choosing the Appropriate Technology for Your Administrative Tasks

WebLogic Server supports a wide range of technologies for performing administrative tasks, including the browser-based WebLogic Remote Console, the Jython-based WebLogic Scripting Tool, WebLogic RESTful management services, and several components for administering and monitoring domains, applications, server life cycle, performance, and more.

Table 2-1 describes common system administration tasks and associated technologies.

**Table 2-1 Choosing the Appropriate Management Technology**

| To do this... | Use this technology... |
|---|---|
| Create domains | The Configuration Wizard guides you through the process of creating or extending a domain for your target environment. See Creating WebLogic Domains Using the Configuration Wizard.<br>To automate the creation of domains, use the WebLogic Scripting Tool, which is a command-line scripting interface based on Jython. See Creating Domains Using WLST Offline in *Understanding the WebLogic Scripting Tool*.<br>Or create domain configuration XML files that conform to the WebLogic Server schema. See Domain Configuration Files in *Understanding Domain Configuration for Oracle WebLogic Server*. |
| Migrate domains from development environments to production environments | Domain Template Builder's `pack` command archives a snapshot of a domain into a JAR file. The `unpack` command expands the archive and creates the necessary start scripts and certain security and configuration files. See *Creating Templates and Domains Using the Pack and Unpack Commands*. |
| Track changes in a domain's configuration | In environments where configuration changes to active domains are allowed, WebLogic Server automatically maintains a versioned archive of configuration files. See Configuration File Archiving in *Understanding Domain Configuration for Oracle WebLogic Server*.<br>To receive real-time notifications that a domain's configuration has been modified, enable the configuration auditing feature. See Configuring the WebLogic Auditing Provider in *Administering Security for Oracle WebLogic Server*.<br>For tightly controlled production environments, configure the run-time domain to be read-only (see Restricting Configuration Changes in *Understanding Domain Configuration for Oracle WebLogic Server*). You can change the read-only setting if you need to roll in changes that have been tested and approved in a staging environment, or you can modify and test your staging environment, and then use a web server to re-route requests from your production environment to the staging environment. |
| Configure connections to databases or other systems | Within individual applications, you can define your own data sources or database connections using JDBC, or connect to external systems using resource adapters. When you deploy such an application, WebLogic Server creates the data sources and connections for you. See:<br>• Configuring WebLogic JDBC Resources in *Administering JDBC Data Sources for Oracle WebLogic Server*<br>• Understanding Resource Adapters in *Developing Resource Adapters for Oracle WebLogic Server*<br>If you have not defined your own data sources or connections within an application, you can use the WebLogic Remote Console, or the WebLogic Scripting Tool to create the resources. See the *Oracle WebLogic Remote Console Online Help* or Using the WebLogic Scripting Tool in *Understanding the WebLogic Scripting Tool*. |
| Manage the server life cycle | The Node Manager is a utility for remote control of Administration Servers and Managed Servers. It runs separately from WebLogic Server and lets you start up and shut down Administration Servers and Managed Servers. While use of Node Manager is optional, it provides additional life cycle benefits if your WebLogic Server environment hosts applications with high availability requirements. See Using Node Manager to Control Servers in the *Administering Node Manager for Oracle WebLogic Server*.<br>To start Administration Servers or Managed Servers without using Node Manager, use the WebLogic Scripting Tool or scripts that WebLogic Server installs. See Starting and Stopping Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*. |

**Table 2-1 (Cont.) Choosing the Appropriate Management Technology**

| To do this... | Use this technology... |
|---|---|
| Configure Coherence Clusters | The WebLogic Remote Console provides a graphical user interface for configuring and managing Coherence clusters; configuring and managing cluster members; and deploying Coherence applications. See the *Oracle WebLogic Remote Console Online Help*.<br>If you prefer a command-line interface, use the WebLogic Scripting Tool. See Using the WebLogic Scripting Tool in *Understanding the WebLogic Scripting Tool*. |
| Modify or add services to an active domain | The WebLogic Remote Console provides a graphical user interface for modifying or adding services to an active domain. See the *Oracle WebLogic Remote Console Online Help*.<br>If you prefer a command-line interface, use the WebLogic Scripting Tool in interactive mode. See Using the WebLogic Scripting Tool in *Understanding the WebLogic Scripting Tool*. |
| Monitor application server services and resources | Monitor the performance of services such as the EJB container, servlet container, and JDBC data sources from the WebLogic Remote Console or through Fusion Middleware Control.<br>Configure policy expressions and actions in the WebLogic Diagnostics Framework to automatically notify administrators of monitoring data events or integrate automated systems through JMX or JMS. See Configuring Policies and Actions in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.<br>If you use SNMP in your operations center, you can enable WebLogic Server to send SNMP notifications for run-time events that you define. See *Monitoring Oracle WebLogic Server with SNMP*. |
| Deploy applications | The WebLogic Remote Console helps guide you through the deployment process. See *Oracle WebLogic Remote Console Online Help*.<br>To automate the deployment of applications, use the WebLogic Scripting Tool. See Deployment Commands in *WLST Command Reference for WebLogic Server*. You can also use the deployment API to write Java programs that deploy applications. See *Deploying Applications with the WebLogic Deployment API*.<br>For information about additional deployment utilities and APIs, see Deployment Tools in *Deploying Applications to Oracle WebLogic Server*. |
| Modify applications in an active domain | To modify the configuration of a deployed application, use a text editor or IDE to modify the deployment descriptor. Then either redeploy the application or use the deployment API to upload the modified deployment descriptor and cause the application container to re-read the deployment descriptor.<br>See *Deploying Applications to Oracle WebLogic Server*. |
| Monitor activity within applications | Determine which data points you want to monitor and then instrument one or more beans to expose this data through JMX. See *Developing Manageable Applications Using JMX for Oracle WebLogic Server*.<br>Alternatively, use the WebLogic Server Diagnostics Service to insert instrumentation code into a running application and monitor its methods or monitor transactions that involve the application. Use this technology to discover the cause of problems that cannot otherwise be discovered by scanning the available monitoring metrics. If you determine that the problem is within your application, you can prevent the problem from recurring by using JMX to expose attributes that indicate the application's health state is degrading. See *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*. |

**Table 2-1 (Cont.) Choosing the Appropriate Management Technology**

| To do this... | Use this technology... |
|---|---|
| Optimize the performance of your application and maintain service-level agreements. | Work Managers configure how your application prioritizes the execution of its work. Based on rules you define and by monitoring actual run-time performance, WebLogic Server can optimize the performance of your application and maintain service-level agreements.<br>See Using Work Managers to Optimize Scheduled Work in *Administering Server Environments for Oracle WebLogic Server*. |
| Configure and secure administration communications | You can separate administration traffic from application traffic in your domain by enabling the administration port. In production environments, separating the two forms of traffic ensures that critical administration operations (starting and stopping servers, changing a server's configuration, and deploying applications) do not compete with high-volume application traffic on the same network connection.<br>The administration port only accepts communications that use SSL, and therefore secures your administrative requests. See Administration Port and Administrative Channel in *Administering Server Environments for Oracle WebLogic Server*. |
| Configure logging and view log files | Many WebLogic Server operations generate logs of their activity. Each server has its own log as well as a standard HTTP access log. These log files can be configured and used in a variety of ways to monitor the health and activity of your servers and applications.<br>By default, WebLogic Server uses the standard JDK logging APIs to filter and write the messages to log files. See Understanding WebLogic Logging Services in *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*. |

# Summary of System Administration Tools and APIs

WebLogic Server includes several of its own standards-based, extensible utilities that you can use to create, manage, and monitor domains. As an alternative, you can also use WebLogic Server's management APIs to create custom management utilities.

Table 2-2 describes the utilities that are included with WebLogic Server.

**Table 2-2 Management Utilities**

| Utility | Description |
|---|---|
| WebLogic Remote Console | The WebLogic Remote Console is a graphical user interface for the administration of Oracle WebLogic Server domains. Use the WebLogic Remote Console to:<br>• Manage WebLogic Server instances and clusters<br>• Create and modify WDT metadata models<br>• Deploy, manage, and monitor applications<br>• Configure security parameters, including managing users, groups, and roles<br>• Configure WebLogic Server services, such as database connectivity (JDBC), and messaging (JMS)<br>Through the WebLogic Remote Console, system administrators can easily perform all WebLogic Server management tasks without having to learn the underlying management architecture. The Administration Server persists changes to attributes in the `config.xml` file for the domain you are managing.<br>See:<br>• Overview of Administration Consoles<br>• *Oracle WebLogic Remote Console Online Help*. |
| WebLogic Scripting Tool | The WebLogic Scripting Tool (WLST) is a command-line scripting interface that you use to manage and monitor active or inactive WebLogic Server domains. The WLST scripting environment is based on the Java scripting interpreter Jython. In addition to WebLogic scripting functions, you can use common features of interpreted languages, including local variables, conditional variables, and flow control statements. You can extend the WebLogic scripting language by following the Jython language syntax. See `http://www.jython.org`.<br>See *Understanding the WebLogic Scripting Tool*. |
| WebLogic Deploy Tooling | WebLogic Deploy Tooling (WDT) simplifies the automation of WebLogic Server domain provisioning and applications deployment.<br>WDT creates a declarative, metadata model that describes the domain, applications, and resources used by applications. This metadata model makes it easy to provision, deploy, and perform domain lifecycle operations in a repeatable fashion.<br>See WebLogic Deploy Tooling. |
| RESTful management resources | WebLogic RESTful management resources provide a comprehensive public interface for configuring, monitoring, deploying and administering WebLogic Server in all supported environments. See About the WLS RESTful Management Interface in *Administering Oracle WebLogic Server with RESTful Management Services*. |
| Configuration Wizard | The Configuration Wizard creates the appropriate directory structure for a WebLogic Server domain, a `config.xml` file, and scripts you can use to start the servers in your domain. The wizard uses templates to create domains, and you can customize these templates to duplicate your own domains.<br>You can also use the Configuration Wizard to add or remove services from an existing, inactive domain.<br>You can run the Configuration Wizard through a graphical user interface (GUI) or in a text-based command-line environment. This command-line environment is called *console mode*. You can also create user-defined domain configuration templates for use with the Configuration Wizard.<br>See *Creating WebLogic Domains Using the Configuration Wizard*. |

**Table 2-2 (Cont.) Management Utilities**

| Utility | Description |
|---|---|
| Domain Template Builder | The Domain Template Builder provides the capability to easily create your own domain templates, to enable, for example, the definition and propagation of a standard domain across a development project, or to enable the distribution of a domain along with an application that has been developed to run on that domain. The templates you create with the Configuration Template Builder are used as input to the Configuration Wizard as the basis for creating a domain that is customized for your target environment. See *Creating Domain Templates Using the Domain Template Builder*. |
| Apache Ant tasks | You can use two Ant tasks provided with WebLogic Server to help you perform common configuration tasks in a development environment. Ant is a Java-based build tool similar to Make. The configuration tasks let you start and stop WebLogic Server instances as well as create and configure WebLogic Server domains. When combined with other WebLogic Ant tasks, you can create powerful build scripts for demonstrating or testing your application with custom domains.<br>See Using Ant Tasks to Configure a WebLogic Server Domain in *Developing Applications for Oracle WebLogic Server*. |
| SNMP Agents | WebLogic Server includes the ability to communicate with enterprise-wide management systems using Simple Network Management Protocol (SNMP). WebLogic Server SNMP agents let you integrate management of WebLogic Servers into an SNMP-compliant management system that gives you a single view of the various software and hardware resources of a complex, distributed system.<br>See *Monitoring Oracle WebLogic Server with SNMP*. |

Table 2-3 describes APIs that you can use to create your own management utilities.

**Table 2-3 Management APIs**

| API | Description |
|---|---|
| JMX | JMX is the Jakarta EE solution for monitoring and managing resources on a network. Like SNMP and other management standards, JMX is a public specification and many vendors of commonly used monitoring products support it.<br>The WebLogic Scripting Tool and other WebLogic Server utilities use the JMX APIs.<br>See *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*. |
| Jakarta Management API | The Jakarta Management APIs (JSR-77) enable a software developer to create a single Java program that can discover and browse resources, such as JDBC connection pools and deployed applications, on any Jakarta EE web application server. The APIs are part of the Jakarta Management Specification, which requires all Jakarta EE web application servers to describe their resources in a standard data model.<br>See *Developing Jakarta Management Applications for Oracle WebLogic Server*. |

**Table 2-3 (Cont.) Management APIs**

| API | Description |
|---|---|
| Deployment API | The WebLogic Server deployment API implements and extends the JSR-88 deployment specification. All WebLogic Server deployment tools, such as the WebLogic Remote Console and wldeploy Ant task, use the deployment API to configure, deploy, and redeploy applications in a domain. You can use the deployment API to build your own WebLogic Server deployment tools, or to integrate WebLogic Server configuration and deployment operations with an existing JSR-88-compliant tool.<br>See *Deploying Applications with the WebLogic Deployment API*. |
| WebLogic Diagnostic Service APIs | The WebLogic Diagnostic Service includes a set of standardized APIs that enable dynamic access and control of diagnostic data, as well as improved monitoring that provides visibility into the server. The interfaces are standardized to facilitate future enhancement and integration of third-party tools, while maintaining the integrity of the server code base. The service is well suited to the server and the server's stack product components and targets operations and administrative staff as primary users.<br>See *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*. |
| Logging APIs | By default, WebLogic Server uses the standard JDK logging APIs to filter and write the messages to log files. See Understanding WebLogic Logging Services in *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*. |

# Roadmap for Administering the WebLogic Server System

The WebLogic Server documentation set includes several introductory, procedural, and reference topics, including examples, that help you understand how to use each of the administration tools and technologies provided by WebLogic Server.

**Table 2-4 Roadmap for Administering the WebLogic Server System**

| Major Task | Subtasks and Additional Information |
|---|---|
| Understanding WebLogic Server system administration | • Overview of WebLogic Server domains<br>• Overview of WebLogic Server clusters<br>• Overview of WebLogic security<br>• Overview of administration consoles<br>• *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*<br>• *Tuning Performance of Oracle WebLogic Server* |
| Installing or upgrading WebLogic Server | • *Installing and Configuring Oracle WebLogic Server and Coherence*<br>• *Creating WebLogic Domains Using the Configuration Wizard*<br>• Oracle Fusion Middleware Supported System Configurations<br>• *What's New in Oracle WebLogic Server*<br>• *Release Notes for Oracle WebLogic Server*<br>• WebLogic Server compatibility<br>• *Upgrading Oracle WebLogic Server* |

**Table 2-4 (Cont.) Roadmap for Administering the WebLogic Server System**

| Major Task | Subtasks and Additional Information |
|---|---|
| Configuring a server environment | • Summary of system administration tools and APIs<br>• Managing configuration changes<br>• *Oracle WebLogic Remote Console Online Help*<br>• *Understanding the WebLogic Scripting Tool*<br>• *Creating Domain Templates Using the Domain Template Builder* |
| Learning about server startup and shutdown | • Overview of starting and stopping servers<br>• Understanding the life cycle of WebLogic Server instances<br>• Server startup command-line reference<br>• Quick reference for starting and stopping servers |
| Starting or stopping a WebLogic Server instance | • Using shell scripts<br>• Using the Remote Console<br>• Using the WebLogic Scripting Tool (WLST)<br>• Using Node Manager to control remote servers<br>• Using the quick reference |
| Configuring Coherence clusters | • Configuring and managing Coherence clusters<br>• *Developing Oracle Coherence Applications for Oracle WebLogic Server*<br>• Securing Oracle Coherence in Oracle WebLogic Server |
| Configuring security | • Overview of WebLogic Server security<br>• *Administering Security for Oracle WebLogic Server*<br>• *Securing a Production Environment for Oracle WebLogic Server*<br>• *Securing Resources Using Roles and Policies for Oracle WebLogic Server* |
| Managing server and network communications | • Configuring network resources<br>• Configuring web server functionality<br>• *Using Oracle WebLogic Server Proxy Plug-Ins* |
| Configuring system resources | • *Administering JDBC Data Sources for Oracle WebLogic Server*<br>• *Administering JMS Resources for Oracle WebLogic Server*<br>• Configuring WebLogic transactions<br>• Configuring the WebLogic Tuxedo Connector<br>• Configuring the persistent store |
| Configuring and deploying applications | • *Deploying Applications to Oracle WebLogic Server*<br>• Configuring web applications<br>• Configuring XML resources<br>• Configuring resource adapters<br>• *Understanding WebLogic Web Services for Oracle WebLogic Server* |
| Monitoring your domain | • *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*<br>• *Monitoring Oracle WebLogic Server with SNMP*<br>• *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*<br>• *Developing Jakarta Management Applications for Oracle WebLogic Server*<br>• Using the Monitoring Dashboard |
| Configuring server environments for high availability | • Understanding cluster architectures<br>• Setting up WebLogic Server clusters<br>• Using session replication across clusters<br>• Using Work Managers to prioritize application execution<br>• Avoiding and managing overload |

**Table 2-4 (Cont.) Roadmap for Administering the WebLogic Server System**

| Major Task | Subtasks and Additional Information |
|---|---|
| Understanding the WebLogic persistent store | • Using the WebLogic persistent store<br>• Create a File Store<br>• Tuning the WebLogic persistent store |
| Troubleshooting | • Viewing the WebLogic Server error message catalog<br>• *Tuning Performance of Oracle WebLogic Server*<br>• Troubleshooting common problems with clustering<br>• *Administering Node Manager for Oracle WebLogic Server* |
| Reference | • *Administration Console Accessibility Notes for Oracle WebLogic Server*<br>• *Command Reference for Oracle WebLogic Server*<br>• *SNMP MIB for Oracle WebLogic Server*<br>• *WLST Command Reference for WebLogic Server*<br>• *MBean Reference for Oracle WebLogic Server* |

3

# Overview of the Administration Console

Use the WebLogic Remote Console for administering Oracle WebLogic Server.

This chapter includes the following topic:

## Using the WebLogic Remote Console

The WebLogic Remote Console is a web browser-based, graphical user interface that you use to manage a WebLogic Server domain. It provides the starting point for essential operations, administration, automation, and management.

The WebLogic Remote Console relies on REST APIs to provide flexibility, enabling it to connect to domains in varied environments: on physical or virtual machines, in containers or Kubernetes pods, in the cloud. Additionally, you can use the WebLogic Remote Console to manage WebLogic Deploy Tooling (WDT) metadata models - meaning you can build domain templates and then manage them within the same environment.

The WebLogic Remote Console is compatible with WebLogic Server 12.2.1.3.0 or later and adjusts its user interface to match the features available to the relevant WebLogic Server release. For detailed information about installing and using the Remote Console, see the Get to Know the Console in the *Oracle WebLogic Remote Console Online Help*.

4

# WebLogic Server Domains

A WebLogic Server domain is a logically related group of WebLogic Server instances, and the resources running on and connected to them, that can be managed as a single administrative unit.

This chapter includes the following topics:

## Understanding Domains

The core of a WebLogic domain consists of the Administration Server, which is the central point from which you configure and manage all resources in the domain.

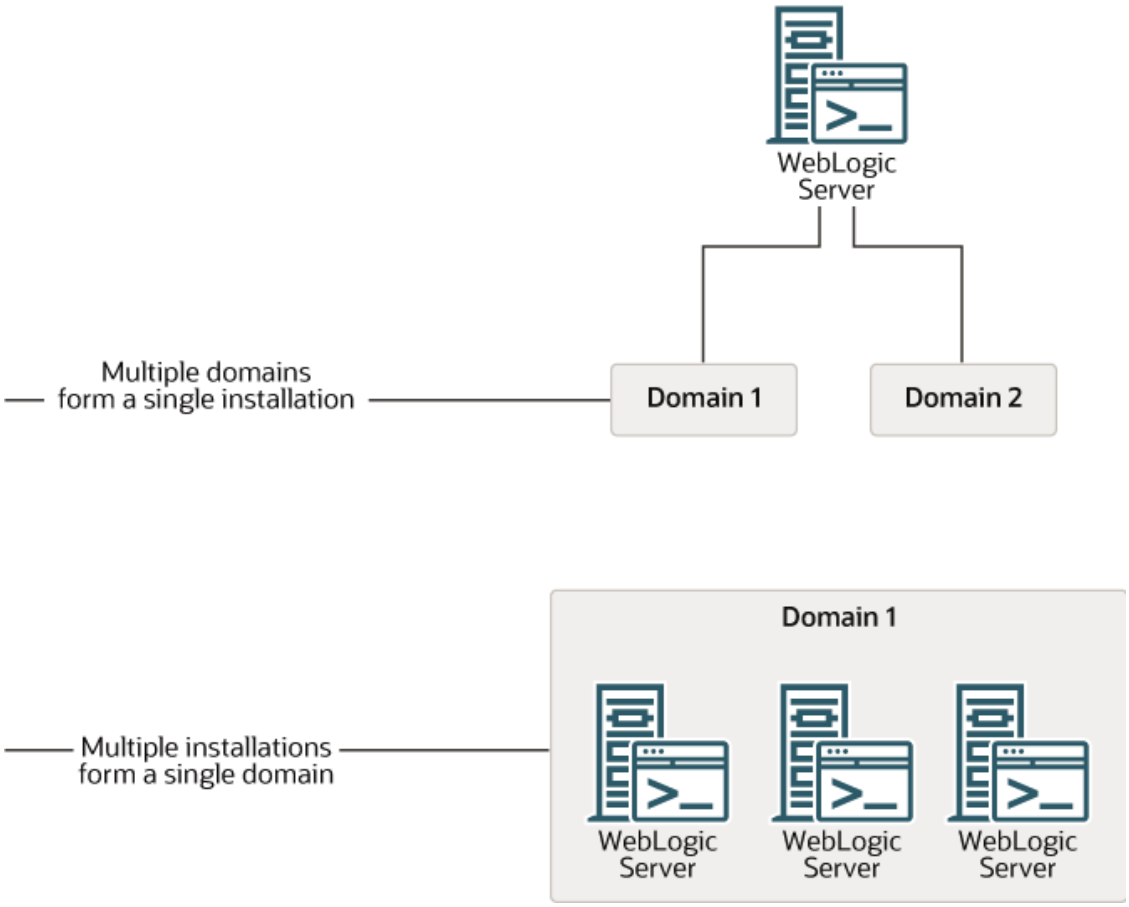
Usually, you configure a domain to include additional WebLogic Server instances called **Managed Servers**. You deploy web applications, EJBs, web services, and other resources onto the Managed Servers and use the **Administration Server** for configuration and management purposes only.

## Organizing Domains

A WebLogic domain is distinct from a WebLogic Server installation. You create and run multiple domains using a single WebLogic Server installation, or you can create a single domain that encompasses multiple installations. How you organize your domains is based on your needs.

Figure 4-1 shows the relationship between a WebLogic Server installation and a WebLogic domain.

**Figure 4-1 Oracle WebLogic Server Installations and Domains**




How you organize your Oracle WebLogic Server installations into domains depends on your business needs. You can define multiple domains based on different system administrators' responsibilities, application boundaries, or geographical locations of the machines on which servers run. Conversely, you might decide to use a single domain to centralize all Oracle WebLogic Server administration activities.

Depending on your particular business needs and system administration practices, you might decide to organize your domains based on criteria such as:

- Logical divisions of applications. For example, you might have one domain devoted to end-user functions such as shopping carts and another domain devoted to back-end accounting applications.
- Physical location. You might establish separate domains for different locations or branches of your business. Each physical location requires its own Oracle WebLogic Server installation.
- Size. You might find that domains organized in small units can be managed more efficiently, perhaps by different system administrators. Contrarily, you might find that maintaining a single domain or a small number of domains makes it easier to maintain a consistent configuration.

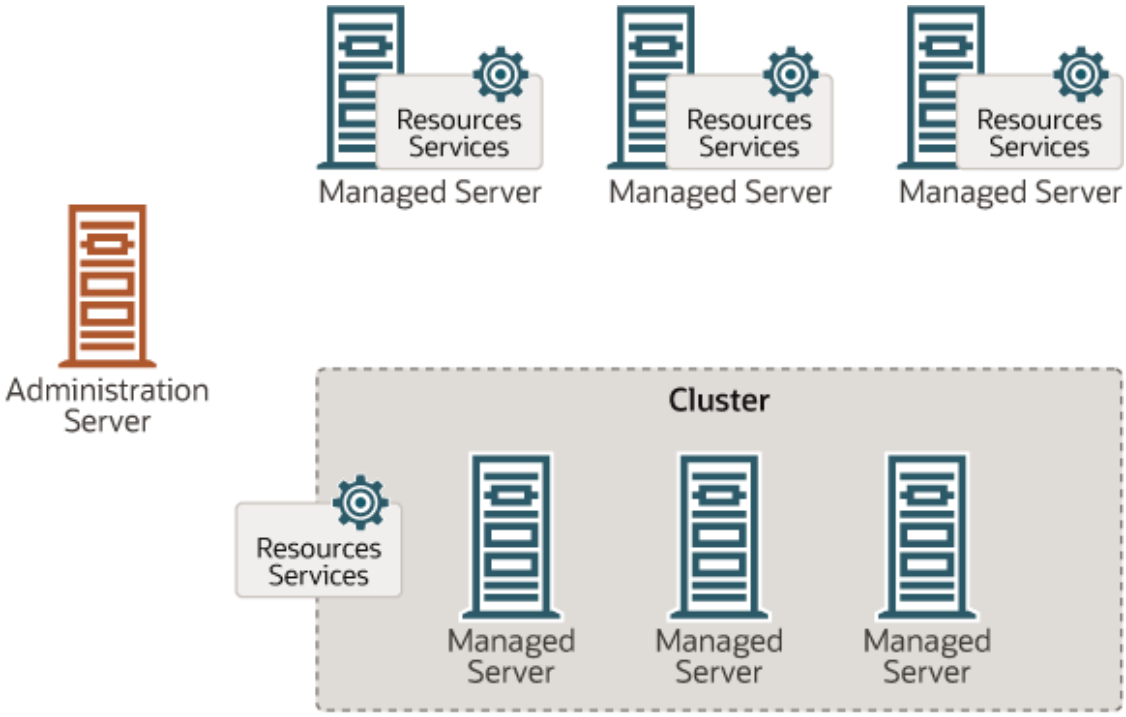
You can create a simple domain that consists of a single server instance. This single instance acts as an Administration Server and hosts the applications that you are developing. Although a single server domain is typically used for development and test environments, this domain type is fully supported for production use and may be appropriate for light-load applications.

The wl_server domain that you can install with Oracle WebLogic Server is an example of this type of domain.

# Contents of a Domain

A domain consists of an Administration Server and optionally one or more Managed Servers. A domain also contains a variety of resources and services used by the those server instances. Figure 4-2 shows a production environment that contains an Administration Server, three stand-alone Managed Servers, and a cluster of three Managed Servers.
Although the scope and purpose of a domain can vary significantly, most Oracle WebLogic Server domains contain the components described in this section.

**Figure 4-2 Contents of a Domain**




The following topics describe the primary components of a domain, which may optionally include managed Coherence servers and Coherence clusters:

## Administration Server

The Administration Server operates as the central control entity for the configuration of the entire domain. It maintains the domain's configuration documents and distributes changes in the configuration documents to Managed Servers. You can also use the Administration Server as a central location from which to monitor all resources in a domain.

To interact with the Administration Server, you can use any of the administration tools listed in Summary of System Administration Tools and APIs. See System Administration for information about modifying the domain's configuration.

Each Oracle WebLogic Server domain must have one server instance that acts as the Administration Server.

For more information about the Administration Server and its role in the Oracle WebLogic Server JMX management system, see System Administration.

## Managed Servers and Managed Server Clusters

Managed Servers host business applications, application components, web services, and their associated resources. To optimize performance, Managed Servers maintain a read-only copy of the domain's configuration document. When a Managed Server starts up, it connects to the domain's Administration Server to synchronize its configuration document with the document that the Administration Server maintains.

For production environments that require increased application performance, throughput, or high availability, you can configure two or more Managed Servers to operate as a **cluster**. A cluster is a collection of multiple Oracle WebLogic Server instances running simultaneously and working together to provide increased scalability and reliability. In a cluster, most resources and services are deployed identically to each Managed Server (as opposed to a single Managed Server), enabling failover and load balancing. A single domain can contain multiple Oracle WebLogic Server clusters, as well as multiple Managed Servers that are not configured as clusters. The key difference between clustered and non-clustered Managed Servers is support for failover and load balancing. These features are available only in a cluster of Managed Servers. For more information about the benefits and capabilities of an Oracle WebLogic Server cluster, see Understanding WebLogic Server Clustering in *Administering Clusters for Oracle WebLogic Server*.

## Managed Coherence Servers and Coherence Clusters

Managed Coherence servers provide in-memory distributed caching for applications. A Managed Server that is configured to be a Coherence cluster member is a managed Coherence server. Coherence is integrated within WebLogic Server as a container subsystem. The use of a container aligns the lifecycle of a Coherence member with the lifecycle of a Managed Server: starting or stopping a server JVM starts and stops a Coherence cluster member.

A domain can contain a single Coherence cluster that can be associated with multiple WebLogic Server clusters. Managed Coherence servers that are part of a WebLogic Server cluster inherit their Coherence settings from the WebLogic Server cluster. WebLogic Server clusters are typically used to setup Coherence tiers that organize managed Coherence servers based on their role in the Coherence cluster.

For details on configuring and managing Coherence clusters, see Configuring and Managing Coherence Clusters in *Administering Clusters for Oracle WebLogic Server*.

## Resources and Services

In addition to the Administration Server and Managed Servers, a domain also contains the resources and services that Managed Servers and deployed applications require.

Managed Servers can use the following resources:

- Machine definitions that identify a particular, physical piece of hardware. A machine definition is used to associate a computer with the Managed Servers it hosts. This information is used by Node Manager in restarting a failed Managed Server, and by a clustered Managed Server in selecting the best location for storing replicated session data. For more information about Node Manager, see Node Manager Overview in the *Administering Node Manager for Oracle WebLogic Server*.

- Network channels that define default ports, protocols, and protocol settings that a Managed Server uses to communicate with clients. After creating a network channel, you can assign it to any number of Managed Servers and clusters in the domain. See Configuring Network Resources in *Administering Server Environments for Oracle WebLogic Server*.
- Virtual hosting, which defines a set of host names to which Oracle WebLogic Server instances (servers) or clusters respond. When you use virtual hosting, you use DNS to specify one or more host names that map to the IP address of a server or cluster. You also specify which web applications are served by each virtual host.

Applications can use the following resources and services:

- Security providers, which are modular components that handle specific aspects of security, such as authentication and authorization.
- Resource adapters, which are system libraries specific to Enterprise Information Systems (EIS) and provide connectivity to an EIS.
- Diagnostics and monitoring services.
- JDBC data sources, which enable applications to connect to databases.
- Mail sessions.
- XML entity caches and registry of XML parsers and transformer factories.
- Messaging services such as JMS servers and store-and-forward services.
- Persistent store, which is a physical repository for storing data, such as persistent JMS messages. It can be either a JDBC-accessible database or a disk-based file.
- Startup classes, which are Java programs that you create to provide custom, system-wide services for your applications.
- Work Managers, which determine how an application prioritizes the execution of its work based on rules you define and by monitoring actual run-time performance. You can create Work Mangers for entire Oracle WebLogic Server domains or for specific application components.
- Work Contexts, which enable applications to pass properties to a remote context without including the properties in a remote call.

# Roadmap for Understanding WebLogic Server Domains

The WebLogic Server documentation set includes several introductory, procedural, and reference topics, including examples, that help you understand how to create, configure, and manage WebLogic domains.

**Table 4-1 Roadmap for Understanding WebLogic Server Domains**

| Major Task | Subtasks and Additional Information |
|---|---|
| Learning more about WebLogic Server domains | • What to do if the Administration Server fails<br>• Domain restrictions<br>• Domain configuration files<br>• Overview of change management<br>• System Administration |

**Table 4-1 (Cont.) Roadmap for Understanding WebLogic Server Domains**

| Major Task | Subtasks and Additional Information |
|---|---|
| Creating domains | • *Creating WebLogic Domains Using the Configuration Wizard*<br>• WebLogic Deploy Tooling Create Domain Tool<br>• Overview of the Configuration Wizard<br>• Extending WebLogic domains<br>• *Creating Templates and Domains Using the Pack and Unpack Commands*<br>• Creating WebLogic domains using WLST offline |
| Configuring domains | • Configuring existing WebLogic domains<br>• *Understanding Domain Configuration for Oracle WebLogic Server*<br>• Managing configuration changes |
| Working with domain templates | • *Creating Domain Templates Using the Domain Template Builder*<br>• Creating and using a domain template (offline) |
| Examples | • WLST Offline Sample Scripts<br>In addition, sample scripts are provided that configure WebLogic domain resources using WLST offline and online on the Oracle Technology Network site.<br>• WebLogic Deploy Tooling Samples |
| Reference | • *Domain Template Reference*<br>• Domain configuration schema<br>• Domain security schema |

5

# WebLogic Server Clustering

The foundation of high availability in WebLogic Server is the cluster. A WebLogic Server cluster is a group of WebLogic Server instances running simultaneously and working together to provide increased scalability and reliability.

This chapter includes the following topics:

## Overview of WebLogic Server Clusters

The server instances that constitute a cluster can run on the same machine, or be distributed across multiple machines. A cluster appears to clients to be a single WebLogic Server instance. You can increase a cluster's capacity by adding additional server instances to the cluster on an existing machine, or you can add machines to the cluster to host the incremental server instances. Each server instance in a cluster must run the same version of WebLogic Server.

## Relationship Between Clusters and Domains

A cluster is part of a particular WebLogic domain. A domain includes one or more WebLogic Server instances. In a domain with multiple server instances, those servers can be clustered, nonclustered, or a combination of clustered and nonclustered instances.

A domain can include multiple clusters. A domain also contains the application components deployed in the domain, and the resources and services required by those application components and the server instances in the domain. Examples of the resources and services used by applications and server instances include machine definitions, optional network channels, connectors, and startup classes.

You can use a variety of criteria for organizing WebLogic Server instances into domains. For instance, you might choose to allocate resources to multiple domains based on logical divisions of the hosted application, geographical considerations, or the number or complexity of the resources under management. For additional information about domains see Understanding Oracle WebLogic Server Domains in *Understanding Domain Configuration for Oracle WebLogic Server*.

In each domain, one WebLogic Server instance acts as the Administration Server—the server instance which configures, manages, and monitors all other server instances and resources in the domain. Each Administration Server manages one domain only. If a domain contains multiple clusters, each cluster in the domain has the same Administration Server. All server instances in a cluster must reside in the same domain; you cannot "split" a cluster over multiple domains. Similarly, you cannot share a configured resource or subsystem between domains.

Clustered WebLogic Server instances behave similarly to nonclustered instances, except that they provide failover and load balancing. The process and tools used to configure clustered WebLogic Server instances are the same as those used to configure nonclustered instances. However, to achieve the load balancing and failover benefits that clustering enables, you must adhere to certain guidelines for cluster configuration.

# Relationship Between Coherence and WebLogic Server Clusters

Similar to WebLogic Server clusters, Coherence clusters consist of multiple managed Coherence server instances that work together to distribute data in-memory to increase application scalability, availability, and performance. However, Coherence clusters use different clustering protocols and are configured separately from WebLogic Server clusters.

With Coherence clusters, a client interacts with the data in a local cache, and the distribution and backup of the data is automatically performed across cluster members.

A WebLogic Server domain can contain a single Coherence cluster. Multiple WebLogic Server clusters can be associated with a Coherence cluster.

For details on configuring and managing Coherence clusters, see Configuring and Managing Coherence Clusters in *Configuring and Managing Coherence Clusters*.

# Benefits of Clustering

Clustering provides two key benefits: scalability and high availability. A WebLogic Server cluster provides the following benefits:

- Scalability

  The capacity of an application deployed on a WebLogic Server cluster can be increased dynamically to meet demand. You can add server instances to a cluster without interruption of service—the application continues to run without impact to clients and end users.

- High availability

  In a WebLogic Server cluster, application processing can continue when a server instance fails. You *cluster* application components by deploying them on multiple server instances in the cluster—so, if a server instance on which a component is running fails, then another server instance on which that component is deployed can continue application processing.

# Key Capabilities of Clusters

A WebLogic cluster has three key capabilities that enable its primary benefits: failover, migration, and load balancing. The key clustering capabilities that enable scalability and high availability include the following:

- Application failover

  If an application component that is doing a particular set of operations becomes unavailable for any reason, then a copy of the failed application component finishes those operations.

- Migration

  WebLogic Server supports automatic and manual migration of a clustered server instance from one machine to another. A Managed Server that can be migrated is referred to as a migratable server. This feature is designed for environments with requirements for high availability.

- Load balancing

  Load balancing is the even distribution of jobs and associated communications across the computing and networking resources in your environment.

# Objects That Can Be Clustered

A clustered application or application component is one that is available on multiple WebLogic Server instances in a cluster. Knowing what objects can be clustered is key to understanding how objects should be deployed throughout your domain. If an object is clustered, failover and load balancing for that object is available. To simplify cluster administration, maintenance, and troubleshooting, make sure that you deploy objects homogeneously; that is, to every server instance in your cluster.
Web applications can consist of different types of objects, including Jakarta Enterprise Beans (EJBs), servlets, and Jakarta Server Pages (JSPs). Each object type has a unique set of behaviors related to control, invocation, and how it functions within an application. For this reason, the methods that WebLogic Server uses to support clustering—and hence to provide load balancing and failover—can vary for different types of objects. The following types of objects can be clustered in a WebLogic Server deployment:

- Servlets
- JSPs
- EJBs
- Remote Method Invocation (RMI) objects
- Jakarta Messaging (JMS) destinations
- Coherence cluster and managed Coherence servers
- Timer services
- Batch applications

# About Dynamic Clusters

Dynamic clusters consist of server instances that can be dynamically scaled up to meet the resource needs of your application. A dynamic cluster uses a single server template to define configuration for a specified number of generated (dynamic) server instances.

When you create a dynamic cluster, the dynamic servers are preconfigured and automatically generated for you, enabling you to easily scale up the number of server instances in your dynamic cluster when you need additional server capacity. You can simply start the dynamic servers without having to first manually configure and add them to the cluster.

If you need additional server instances on top of the number you originally specified, you can increase the maximum number of servers instances (dynamic) in the dynamic cluster configuration or manually add configured server instances to the dynamic cluster. A dynamic cluster that contains both dynamic and configured server instances is called a mixed cluster.

The following table defines terminology associated with dynamic clusters:

| Term | Definition |
|---|---|
| dynamic cluster | A cluster that contains one or more generated (dynamic) server instances that are based on a single shared server template. |
| configured cluster | A cluster in which you manually configure and add each server instance. |
| dynamic server | A server instance that is generated by WebLogic Server when creating a dynamic cluster. Configuration is based on a shared server template. |
| configured server | A server instance for which you manually configure attributes. |

| Term | Definition |
|---|---|
| mixed cluster | A cluster that contains both dynamic and configured server instances. |
| server template | A prototype server definition that contains common, non-default settings and attributes that can be assigned to a set of server instances, which then inherit the template configuration. For dynamic clusters, the server template is used to generate the dynamic servers. See Server Templates in *Understanding Domain Configuration for Oracle WebLogic Server*. |

For more information about dynamic clusters, see Dynamic Clusters in *Administering Clusters for Oracle WebLogic Server*.

# Roadmap for Clustering in WebLogic Server

The WebLogic Server documentation set includes several introductory, procedural, and reference topics, including examples, that help you understand how to configure and manage WebLogic clusters.

**Table 5-1 Roadmap for Clustering in WebLogic Server**

| Major Task | Subtasks and Additional Information |
|---|---|
| Learning more about WebLogic Server clustering | • Clustering servlets and JSPs<br>• Clustering EJBs and RMI objects<br>• JMS and clustering<br>• Coherence clustering<br>• Dynamic clusters |
| Configuring a cluster | • Understanding cluster configuration<br>• Communications in a cluster<br>• Cluster architectures<br>• Setting up WebLogic Server clusters<br>• Clustering best practices<br>• Setting up Coherence clusters |
| Configuring elasticity for a dynamic cluster | • Performing on-demand scaling<br>• Configuring elastic actions<br>• Configuring calendar-based scaling<br>• Configuring policy-based scaling |
| Learning more about load balancing and failover in a cluster | • Load balancing in a cluster<br>• Failover and replication in a cluster<br>• Configuring BIG-IP hardware with clusters<br>• Configuring F5 load balancers for MAN/WAN failover<br>• Configuring Radware load balancers for MAN/WAN failover |
| Migrating servers and services in a cluster | • Whole server migration<br>• Service migration |
| Troubleshooting | • Troubleshooting common problems<br>• Troubleshooting multicast configuration |
| Reference | • The WebLogic cluster API |

6

# Developing Applications in WebLogic Server

WebLogic Server implements Jakarta Platform, Enterprise Edition (Jakarta EE) Version 9.1 technologies. Jakarta EE is the standard platform for developing multitier enterprise applications based on the Java programming language.

This chapter includes the following topics:

## WebLogic Server and the Jakarta EE Platform

With Jakarta EE, development of Jakarta enterprise applications has never been easier or faster. The aim of the Jakarta EE platform is to provide developers with a powerful set of APIs while shortening development time, reducing application complexity, and improving application performance. The technologies that make up Jakarta EE were developed collaboratively by several software vendors. For background information on Jakarta EE, refer to https://jakarta.ee/learn/.
Starting in earlier versions and continuing in Jakarta EE 9.1, the focus has been ease of development. There is less code to write – much of the boilerplate code has been removed, defaults are used whenever possible, and annotations are used extensively to reduce the need for deployment descriptors.

- EJB 4.0 provides simplified programming and packaging model changes. The mandatory use of Java interfaces from previous versions has been removed, allowing plain old Java objects to be annotated and used as EJB components. The simplification is further enhanced through the ability to place EJB modules directly inside web applications, removing the need to produce archives to store the web and EJB components and combine them together in an EAR file.
- Jakarta EE 9.1 continues the focus on modern web applications and broadening the range of such applications. The key goals of the Jakarta EE platform are to modernize the infrastructure for enterprise Java for the cloud and microservices environments, emphasize HTML5 and HTTP/2 support, and enhance the ease of development through new Contexts and Dependency Injection (CDI) features, and further enhance the security and reliability of the platform.
- Constructing web applications is made easier with Jakarta Server Faces (JSF) technology and the JSP Standard Tag Library (JSTL). Jakarta EE 9.1 supports rich thin-client technologies such as AJAX, for building applications for Web 2.0.

WebLogic Server Jakarta EE applications are based on standardized, modular components. WebLogic Server provides a complete set of services for those modules and handles many details of application behavior automatically, without requiring programming. Jakarta EE defines module behaviors and packaging in a generic, portable way, postponing runtime configuration until the module is actually deployed on an application server.

Jakarta EE includes deployment specifications for web applications, EJB modules, web services, enterprise applications, client applications, and connectors. Jakarta EE does not specify *how* an application is deployed on the target server—only how a standard module or application is packaged. For each module type, the specifications define the files required and their location in the directory structure.

Jakarta EE is platform independent, so you can edit and compile code on any platform, and test your applications on development WebLogic Servers running on other platforms. For example, it is common to develop WebLogic Server applications on a PC running Windows or Linux, regardless of the platform where the application is ultimately deployed.

For more information, refer to the Jakarta EE specification at: https://jakarta.ee/specifications/platform/9.1/.

# Overview of Jakarta EE Applications and Modules

At runtime, a Jakarta EE application is a type of module. A WebLogic Server Jakarta EE application consists of one of the following modules or applications running on WebLogic Server:

- Web application modules—HTML pages, servlets, JavaServer Pages, and related files. See Web Application Modules in *Developing Applications for Oracle WebLogic Server*.
- Jakarta Enterprise Beans (EJB) modules—entity beans, session beans, and message-driven beans. See Jakarta Enterprise Bean Modules in *Developing Applications for Oracle WebLogic Server*.
- Connector modules—resource adapters. See Connector Modules in *Developing Applications for Oracle WebLogic Server*.
- Enterprise applications—web application modules, EJB modules, resource adapters and web services packaged into an application. See Enterprise Applications in *Developing Applications for Oracle WebLogic Server*.
- Web services—See WebLogic Web Services in *Developing Applications for Oracle WebLogic Server*.

A WebLogic application can also include the following WebLogic-specific modules:

- JDBC and JMS modules—See JMS and JDBC Modules in *Developing Applications for Oracle WebLogic Server*.
- Coherence Grid modules—See Packaging Coherence Applications in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.
- WebLogic Diagnostic FrameWork (WLDF) modules—See WebLogic Diagnostic Framework Modules in *Developing Applications for Oracle WebLogic Server*.

# Roadmap for Developing Applications in WebLogic Server

The WebLogic Server documentation set includes several introductory, procedural, and reference topics, including examples, that help you understand how to develop Jakarta EE applications on WebLogic Server.

**Table 6-1 Roadmap for Developing Applications in WebLogic Server**

| Major Task | Subtasks and Additional Information |
|---|---|
| Learning more about application development | • XML deployment descriptors<br>• Deployment plans<br>• Best practices for developing WebLogic Server applications<br>• Understanding application life cycle events<br>• Understanding production redeployment<br>• Understanding WebLogic Server application classloading<br>• Overview of shared Jakarta EE libraries and optional packages |

**Table 6-1 (Cont.) Roadmap for Developing Applications in WebLogic Server**

| Major Task | Subtasks and Additional Information |
|---|---|
| Setting up your development environment | • Starting and stopping WebLogic Server<br>• Use the "split development directory" to develop your applications |
| Designing your application | • Using shared Jakarta EE libraries and optional packages to share code among deployed applications<br>• Programming JSF and JSTL applications<br>• Using life cycle listeners<br>• Using the HTTP publish-subscribe server<br>• Using Coherence to cache data<br>• Using Coherence to cache HTTP session data<br>• *Developing Applications with the WebLogic Security Service*<br>• Internationalize or localize your application<br>• Using threads in WebLogic Server<br>• Using WebSockets in WebLogic Server<br>• *Adding WebLogic Logging Services to Applications Deployed on Oracle WebLogic Server*<br>• *Developing Standalone Clients for Oracle WebLogic Server*<br>• Designing manageable applications |
| Building your application | • *Developing Applications for Oracle WebLogic Server*<br>• Deploying your "split development directory" application on WebLogic Server<br>• Using Ant tasks to compile Java code |
| Using development tools | • Development software<br>• Ant<br>• *Oracle WebLogic Remote Console Online Help*<br>• *Command Reference for Oracle WebLogic Server*<br>• *Creating WebLogic Domains Using the Configuration Wizard*<br>• *Creating Domain Templates Using the Domain Template Builder*<br>• *Understanding the WebLogic Scripting Tool* |
| Moving your application to a production environment | • Preparing your application or module for deployment<br>• Configuring your application for production deployment<br>• Updating your deployed application (production redeployment) |

**Table 6-1 (Cont.) Roadmap for Developing Applications in WebLogic Server**

| Major Task | Subtasks and Additional Information |
|---|---|
| Application examples | **Note**<br>The WebLogic Server code examples include a naming convention, which indicates the path names where the samples are located. Jakarta EE 8 Examples, Java EE 7 Examples, and Java EE 6 Examples refers to the names of the folders where these examples are installed and not the version of Java EE or Jakarta EE that the examples support. All the WebLogic Server examples remain relevant for developing WebLogic Server 15.1.1.0.0 applications.<br>• Jakarta EE 8 Examples<br>• Java EE 7 Examples<br>• Java EE 6 Examples<br>• Additional API Examples<br>• Avitek Medical Records<br>A complete and functional Jakarta EE application including source code. The MedRec (Spring) sample application demonstrates Spring 3.0.x application development practices. |
| Jakarta EE API programming guides | • *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*<br>• *Developing Manageable Applications Using JMX for Oracle WebLogic Server*<br>• *Developing Security Providers for Oracle WebLogic Server*<br>• *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*<br>• *Developing Jakarta Management Applications for Oracle WebLogic Server*<br>• *Developing Enterprise JavaBeans for Oracle WebLogic Server*<br>• *Developing JDBC Applications for Oracle WebLogic Server*<br>• *Developing JMS Applications for Oracle WebLogic Server*<br>• *Developing JNDI Applications for Oracle WebLogic Server*<br>• *Developing JTA Applications for Oracle WebLogic Server*<br>• *Developing Resource Adapters for Oracle WebLogic Server*<br>• *Developing RMI Applications for Oracle WebLogic Server*<br>• *Developing XML Applications for Oracle WebLogic Server*<br>• *Developing Standalone Clients for Oracle WebLogic Server*<br>• *Deploying Applications with the WebLogic Deployment API*<br>• *Developing JSP Tag Extensions for Oracle WebLogic Server*<br>• *Developing Applications with the WebLogic Security Service*<br>• *Developing Jakarta XML Web Services for Oracle WebLogic Server*<br>• *Developing CommonJ Applications for Oracle WebLogic Server*<br>• *Adding WebLogic Logging Services to Applications Deployed on Oracle WebLogic Server*<br>• *Administering Clusters for Oracle WebLogic Server*<br>• *Developing Oracle WebLogic Tuxedo Connector Applications for Oracle WebLogic Server* |

**Table 6-1 (Cont.) Roadmap for Developing Applications in WebLogic Server**

| Major Task | Subtasks and Additional Information |
|---|---|
| Javadoc and API reference | • Jakarta EE Platform 9.1<br>• Java Platform, Standard Edition (Java SE) Version 17<br>• Java Platform, Standard Edition (Java SE) Version 21<br>• *JMS C API Reference for Oracle WebLogic Server*<br>• *Java API Reference for Oracle WebLogic Server*<br>• *Microsoft .NET Messaging API for Oracle WebLogic Server* |
| General reference | • XML deployment descriptors<br>• WebLogic JSP cache, process, and repeat tags<br>• WebLogic JSP form validation tags<br>• *Command Reference for Oracle WebLogic Server*<br>• *MBean Reference for Oracle WebLogic Server*<br>• *WebLogic Server Error Message Catalog* |

7

# Deploying Applications in WebLogic Server

Application deployment refers to the process of making an application or module available for processing client requests in a WebLogic Server domain.

This chapter includes the following topics:

## Overview of the Deployment Process

Application deployment encompasses several discrete tasks, such as preparing and configuring applications for deployment, exporting and redeploying applications to new environments, and managing deployed applications.

See the following topics in *Deploying Applications to Oracle WebLogic Server*:

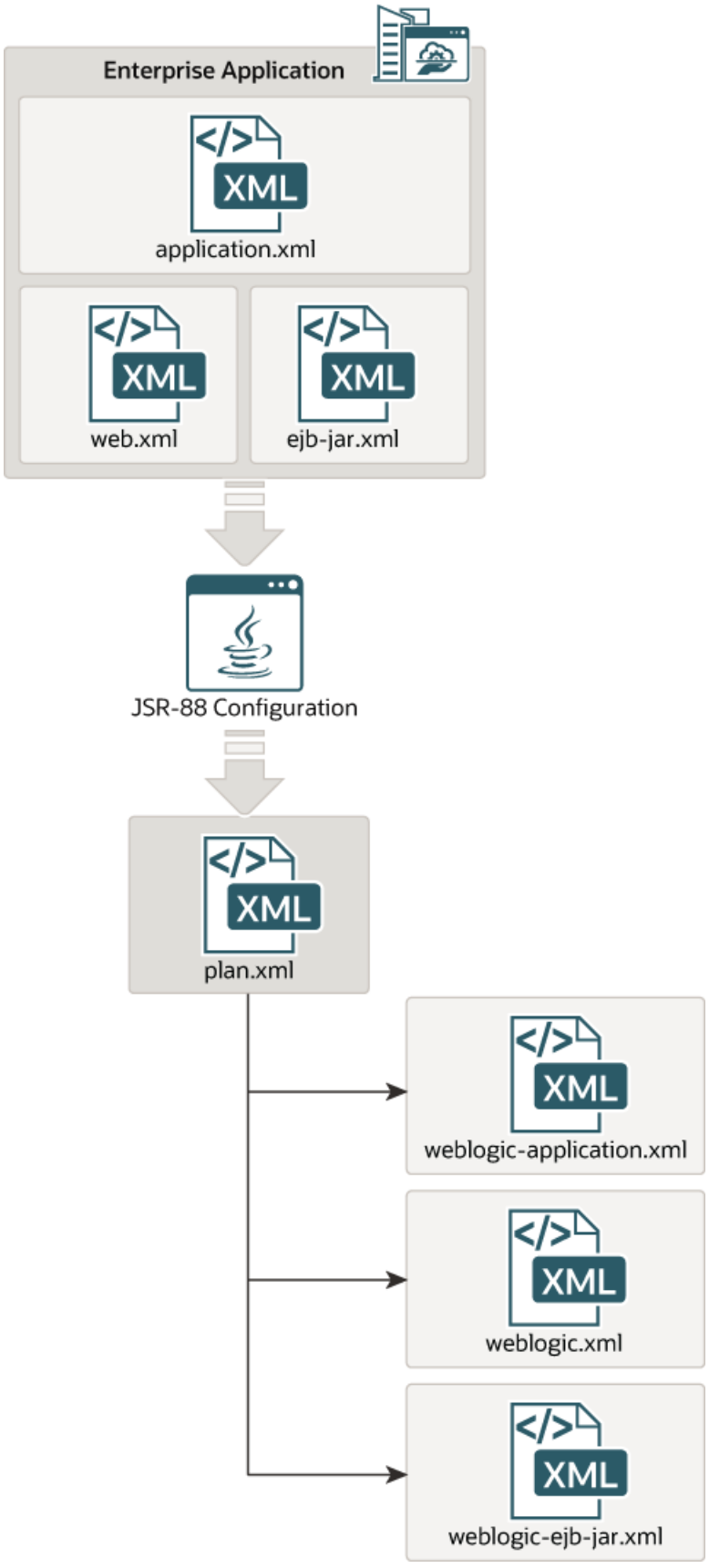
- Preparing Applications and Modules for Deployment
- Configuring Applications for Production Deployment
- Exporting an Application for Deployment to New Environments
- Deploying Applications and Modules with weblogic.Deployer
- Redeploying Applications in a Production Environment
- Managing Deployed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

## Jakarta EE Deployment Implementation

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.

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 7-1.

**Figure 7-1 Configuring Applications with the Jakarta EE Deployment API**



WebLogic Server deployment descriptors are generated as needed to store WebLogic Server configuration data.

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 in *Deploying Applications to Oracle WebLogic Server*.

# Fast Track Deployment Guide

To support application development environments, WebLogic Server provides a robust set of utilities and tools that you can use to deploy your applications quickly. The following topics provide basic instructions for quickly deploying Jakarta EE applications and modules, JSP and HTML files, and Coherence modules. They also provide pointers to tools for system administrators. These deployment procedures are recommended for use in development environments only; the procedures are not recommended for use in production environments. For additional information about developing and deploying applications on WebLogic Server, see *Developing Applications for Oracle WebLogic Server* and *Deploying Applications to Oracle WebLogic Server*.
Complete *Installing and Configuring Oracle WebLogic Server and Coherence* before using the Fast Track procedures described in the following sections:

## Jakarta EE Deployment

To deploy a Jakarta EE application or module:

1. Make sure that the Jakarta EE application or module does not require additional resources such as named JDBC data sources or JMS queues. If the application requires external resources, you must configure them in the target WebLogic Server domain before deploying the application.
2. Copy the archive file or exploded archive directory for the Jakarta EE application or module into the `/autodeploy` directory of the examples server domain directory, *`ORACLE_HOME`*`/user_projects/domains/<wls_examples>/autodeploy`.
3. Start the Examples WebLogic Server instance.
4. Access the application using either a Jakarta client or the configured URI for the application.

### Auto-Deployment

When running in development mode, WebLogic Server automatically deploys applications copied into the `/autodeploy` subdirectory of the domain directory. Auto-deployment is a simple and quick method of deploying an application for testing or evaluation. See Auto-Deploying Applications in Development Domains in *Deploying Applications to Oracle WebLogic Server*.

## System Administrator Tools

System Administrators can use the following tools to get started:

- WebLogic Remote Console

  The WebLogic Remote Console is a browser-based web application that allows you to configure and monitor your WebLogic Server domain, server instances, and running applications and their associated resources. You can also use the WebLogic Remote Console to create new server instances and clusters and tune application descriptors. See *Oracle WebLogic Remote Console Online Help*.

- Configuration Wizard

  Use the WebLogic Server Configuration Wizard to create new domains, and to create templates for automating domain configuration. See *Creating WebLogic Domains Using the Configuration Wizard*.

## JSP/HTML Deployment

To deploy a simple JSP or HTML file:

1. Make sure your JSP file does not reference a tag library or other external resources—such resources require additional deployment steps that are beyond the scope of these Fast Track procedures. HTML files do not have this restriction.
2. Copy your JSP or HTML file into the *EXAMPLES_HOME*`/wl_server/examples/build/mainWebApp` directory, where *EXAMPLES_HOME* represents the directory in which the WebLogic Server code examples are configured. By default, this directory is *ORACLE_HOME*`/wlserver/samples/server`.
3. Start the Examples WebLogic Server instance.
4. In a web browser, request the JSP or HTML file using the following URL:

   `http://localhost:port/myFile`

   where:

   `localhost` is the host name of the machine running WebLogic Server.

   `port` is the port number where WebLogic Server is listening for requests (7001 by default).

   `myFile` is the full name, including the .jsp or .html extension, of the JSP or HTML file you copied in step 2.

The JSP or HTML file has been automatically deployed from a directory preconfigured to target the Examples Server. `mainWebApp` is deployed by default and you can place your own JSP and HTML files into the `mainWebApp` exploded directory in order to quickly view or test them.

## Coherence Deployment

WebLogic Server supports the deployment of Coherence applications that are packaged as Grid ARchive (GAR) modules. GAR modules contain the artifacts that are required for a Coherence application. GAR modules are deployed as standalone modules, packaged within enterprise applications, and as shared libraries. For details on packaging and deploying Coherence applications, see Deploying Coherence Applications in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

# Roadmap for Deploying Applications in WebLogic Server

The WebLogic Server documentation set includes several introductory, procedural, and reference topics, including examples, that help you understand how to deploy applications in the WebLogic Server environment.

**Table 7-1 Roadmap for Deploying Applications in WebLogic Server**

| Major Task | Subtasks and Additional Information |
|---|---|
| Learning more about application deployment | • Deployment terminology<br>• Jakarta EE deployment implementation<br>• WebLogic Server deployment features<br>• Understanding the deployment configuration process<br>• Overview of the export process<br>• Best practices for deploying applications |
| Packaging applications | • Preparing applications and modules for deployment<br>• Archive file and exploded archive deployments<br>• Using the wlpackage Ant task<br>• Preparing Coherence applications for deployment |
| Using deployment tools | • Overview of deployment tasks<br>• weblogic.Deployer utility<br>• WebLogic.Plan generator command-line reference<br>• WebLogic Maven plug-in for deployment<br>• wldeploy Ant task |
| Advanced topics | • Overview of common deployment scenarios<br>• Configuring applications for deployment<br>• Redeploying a production application<br>• *Deploying Applications with the WebLogic Deployment API*<br>• Exporting an application for deployment to new environments<br>• Distributing an application to a production environment<br>• Changing the deployment order<br>• Taking an application offline<br>• Managing deployed applications |
| Reference | • Understanding the WebLogic deployment API |

8

# WebLogic Server Data Sources

In WebLogic Server, you can configure database connectivity by configuring JDBC data sources and then targeting or deploying those resources to servers or clusters in your WebLogic domain.

This chapter includes the following topics:

## Understanding JDBC Data Sources

A data source is a pool of database connections that are created when the data source instance is created, which can occur when the data source is deployed, when it is targeted, or when the host WebLogic Server instance is started.

Oracle WebLogic Server provides five types of data sources:

- Generic data sources—Generic data sources and their connection pools provide connection management processes that help keep your system running efficiently. You can set options in the data source to suit your applications and your environment.
- Active GridLink data sources—An event-based data source that adaptively responds to state changes in an Oracle RAC instance.
- Multi data sources—An abstraction around a group of generic data sources that provides load balancing or failover processing.
- Proxy data sources—Data sources that provide the ability to switch between databases in a WebLogic Server Multitenant environment.
- Universal Connection Pool (UCP) data sources—Data sources provided as an option for users who wish to use Oracle Universal Connection Pooling (UCP) to connect to Oracle Databases. UCP provides an alternative connection pooling technology to Oracle WebLogic Server connection pooling.

WebLogic Server also supports Jakarta EE `DataSource` objects, which can be programmatically defined for a more flexible and portable method of database connectivity. For more information on Jakarta EE `DataSource` objects, see Using DataSource Resource Definitions in *Developing JDBC Applications for Oracle WebLogic Server*.

## Understanding Generic Data Sources

Generic data sources and their connection pools provide database access and database connection management processes that help keep your system running efficiently. Each generic data source contains a pool of database connections that are created when the data source is created and at server startup. Applications reserve a database connection from the data source by looking up the data source on the JNDI tree or in the local application context and then calling `getConnection()`. When finished with the connection, the application should call `connection.close()` as early as possible, which returns the database connection to the pool for other applications to use.

# Understanding Active GridLink Data Sources

A single Active GridLink (AGL) data source provides connectivity between WebLogic Server and an Oracle Database service, which may include multiple Oracle RAC clusters. An AGL data source uses the Oracle Notification Service (ONS) to adaptively respond to state changes in an Oracle RAC instance. An Oracle Database service represents a workload with common attributes that enables administrators to manage the workload as a single entity. You scale the number of AGL data sources as the number of services increases in the database, independent of the number of nodes in the cluster.
An AGL data source includes the features of generic data sources plus the following support for Oracle RAC as described in *Administering JDBC Data Sources for Oracle WebLogic Server*:

- Fast Connection Failover
- Runtime Connection Load Balancing
- Generic Data Source Handling for Oracle RAC Outages
- GridLink Affinity
- SCAN Addresses
- Secure Communication using Oracle Wallet

# Understanding JDBC Multi Data Sources

Conceptually, a multi data source can be regarded as a pool of generic data sources. Multi data sources are best used for failover or load balancing between nodes of a highly available database system, such as redundant databases or Oracle Real Application Clusters (Oracle RAC). A multi data source is bound to the JNDI tree or local application context, in the same way that generic data sources are bound to the JNDI tree. Applications look up a multi data source on the JNDI tree or in the local application context (`java:comp/env`), just as they do for data sources, and then request a database connection. The multi data source determines the data source to use that can satisfy the request depending upon the algorithm selected in the multi data source configuration: load balancing or failover.

# Understanding Universal Connection Pool Data Sources

A Universal Connection Pool (UCP) data source enables the use of Oracle Universal Connection Pooling (UCP) for connecting to Oracle Database. A UCP data source is available as an option for using UCP, which is an alternative connection pooling technology to WebLogic Server connection pooling.

> **Note**
>
> Oracle generally recommends the use of Active GridLink data sources, multi data sources, or generic data sources, and also the Oracle WebLogic Server connection pooling included in these data source implementations to establish connectivity with Oracle Database.

The implementations of UCP data sources are loosely coupled, allowing the swapping of the `ucp.jar` file to support the use of new UCP features by the applications. UCP data sources are

not supported in an application-scoped, application-packaged, or standalone module environment. See Using Universal Connection Pool Data Sources in *Administering JDBC Data Sources for Oracle WebLogic Server*.

# Roadmap for WebLogic Server Data Sources

The WebLogic Server documentation set includes several introductory, procedural, and reference topics, including examples, that help you understand how to configure and use WebLogic Server data sources.

**Table 8-1 Roadmap for WebLogic Server Data Sources**

| Major Task | Subtasks and Additional Information |
|---|---|
| Learning more about WebLogic Server data source | • About WebLogic JDBC resources<br>• JDBC resources<br>• JMX and WLST access for JDBCA resources<br>• Overview of clustered JDBC resources<br>• Multi data source features<br>• Using WebLogic JDBC in an application |
| Configuring JDBC | • Configuring JDBC data sources<br>• Using GridLink data sources<br>• Configuring JDBC multi data sources<br>• Advanced configuration for Oracle drivers<br>• JDBC data source transaction options<br>• Using roles and policies to secure JDBC data sources |
| Jakarta EE DataSources | • Using DataSource resource definitions |
| Managing JDBC | • Managing data sources<br>• Monitoring data sources<br>• Monitoring GridLink JDBC resources |
| Performance and tuning | • Tuning JDBC applications<br>• Tuning data source connection pools |
| Using WebLogic Server with Oracle RAC | • Using WebLogic Server with Oracle RAC<br>• Using multi data sources with Oracle RAC<br>• Using fast connection failover with Oracle RAC |
| Using JDBC drivers | • Overview of third-party JDBC drivers<br>• Derby<br>Derby is an all-Java DBMS product included in the WebLogic Server distribution that is intended solely to support demonstration of WebLogic Server examples. Documentation is not shipped with the product; it is available at http://db.apache.org/derby/manuals/index.html. For more information about Derby, see http://db.apache.org/derby. |

9

# WebLogic Server Messaging

Jakarta Messaging (JMS) is a standard API for accessing enterprise messaging systems. JMS simplifies application development by providing a standard interface for creating, sending, and receiving messages.

This chapter includes the following topics:

## Overview of JMS and WebLogic Server

The WebLogic Server implementation of JMS is an enterprise-class messaging system that is tightly integrated into the WebLogic Server platform and fully supports the JMS 2.0 Specification. The JMS 2.0 Specification is available at http://www.oracle.com/technetwork/java/jms/index.html. WebLogic JMS provides numerous WebLogic JMS Extensions that go beyond the standard JMS APIs.

## Jakarta Messaging

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

The JMS is a standard API for accessing enterprise messaging systems that is implemented by industry messaging providers. Specifically, JMS:

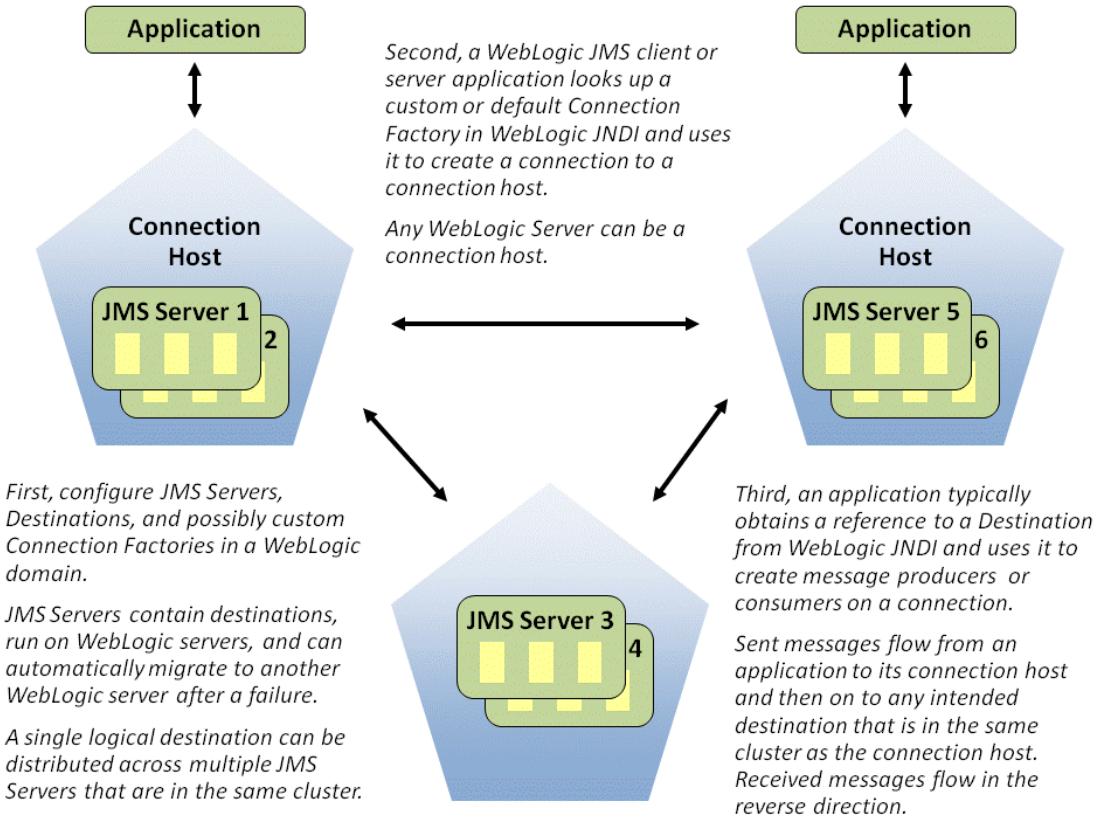
- Enables Jakarta applications that share a messaging system to exchange messages
- Simplifies application development by providing a standard interface for creating, sending, and receiving messages

WebLogic JMS supports both client and server applications; in addition to Java, it has client libraries for C APIs and Microsoft .NET. WebLogic JMS accepts messages from *producer* applications and delivers them to *consumer* applications. For more information on JMS API programming with WebLogic Server, see *Developing JMS Applications for Oracle WebLogic Server*. For information about JMS API programming for WebLogic Server hosted consumer applications, see *Developing Message-Driven Beans for Oracle WebLogic Server*.

#### WebLogic JMS Message Workflow

The following figure illustrates the WebLogic message workflow.

**Figure 9-1 WebLogic JMS Message Workflow**




For information on the major components of the WebLogic JMS architecture, see WebLogic JMS Architecture in *Developing JMS Applications for Oracle WebLogic Server*.

# Roadmap for WebLogic Server Messaging

The WebLogic Server documentation set includes several introductory, procedural, and reference topics, including examples, that help you understand how to develop and administer WebLogic JMS resources.

**Table 9-1 Roadmap for WebLogic Server Messaging**

| Major Task | Subtasks and Additional Information |
|---|---|
| Learning more about WebLogic Server messaging | • WebLogic JMS architecture and environment<br>• JMS configuration resources<br>• Overview of JMS servers<br>• Overview of JMS modules<br>• Environment-related system resources for WebLogic JMS<br>• Understanding the messaging models<br>• Understanding the JMS API<br>• Value-added public JMS API extensions |

**Table 9-1 (Cont.) Roadmap for WebLogic Server Messaging**

| Major Task | Subtasks and Additional Information |
|---|---|
| Getting started with WebLogic JMS | • Overview of JMS programming<br>• Best practices for JMS beginners and advanced users<br>• Developing a basic JMS application<br>• Overview of JMS resource configuration<br>• Value-added WebLogic Server JMS features<br>• Integrating remote and foreign JMS providers<br>• Sample Applications and Code Examples<br>• Troubleshooting WebLogic JMS |
| Using new WebLogic JMS features | • Developing advanced pub/sub applications<br>• Interoperating with Oracle advanced queueing<br>• *Developing JMS .NET Client Applications for Oracle WebLogic Server* |
| Programming WebLogic messaging | • *Developing JMS Applications for Oracle WebLogic Server*<br>• Developing advanced pub/sub applications<br>• *Developing Message-Driven Beans for Oracle WebLogic Server* |
| Understanding clients for WebLogic messaging | • Understanding JMS clients<br>• WebLogic Server client types and features |
| Configuring WebLogic messaging | • Best practices for JMS beginners and advanced users<br>• *Administering JMS Resources for Oracle WebLogic Server*<br>• Integrating remote and foreign JMS providers<br>• *Administering the Store-and-Forward Service for Oracle WebLogic Server*<br>• *Administering the WebLogic Messaging Bridge for Oracle WebLogic Server*<br>• *Administering the WebLogic Persistent Store* |
| Using the WebLogic Remote Console to configure WebLogic messaging | • Create a JMS Server<br>• Create a JMS System Module<br>• Create a Store-and-Forward Agent<br>• Create a Messaging Bridge Instance |
| Performance and tuning | • Tuning WebLogic JMS<br>• Tuning WebLogic JMS store-and-forward<br>• Tuning WebLogic messaging bridge<br>• Tuning message-driven beans<br>• Tuning logging last resource<br>• Tuning the WebLogic Persistent Store |
| Reference | • Javadoc for WebLogic JMS extensions<br>• MBean reference<br>• JMS schema<br>• Jakarta Messaging Specification<br>• *JMS C API Reference for Oracle WebLogic Server* |

10

# Understanding WebLogic Server Security

WebLogic Server includes a security architecture that provides a unique and secure foundation for applications. By taking advantage of the security features in WebLogic Server, enterprises benefit from a comprehensive, flexible security infrastructure designed to address the security challenges of making applications available on the web.

This chapter includes the following topics:

> **Note**
>
> For a complete checklist of critical tasks for locking down WebLogic Server, including specific tasks recommended for configuring a secure domain, securing the network, files, and databases used by WebLogic Server, see *Securing a Production Environment for Oracle WebLogic Server*.

## Jakarta EE Security API Support in WebLogic Server

WebLogic Server 15.1.1.0.0 provides full support for the Jakarta EE Security API 2.0.

The Jakarta EE Security API defines portable authentication mechanisms (such as `HttpAuthenticationMechanism` and `IdentityStore`), and an access point for programmatic security using the `SecurityContext` interface. These portable, plug-in authentication and identity store interfaces provide an advantage over container-provided implementations because they allow the application to control the authentication process and the identity stores used for that authentication in a standard and portable way. Bundling the security configuration in the application, instead of configuring it externally, improves the management of the application's lifecycle, especially in a world of microservices that are distributed in containers. You can use the built-in implementations of these APIs, or define custom implementations.

In WebLogic Server, these authentication mechanisms are supported in the web container, and the `SecurityContext` interfaces are supported in the Servlet and EJB containers. WebLogic Server also supports the requirement in the Security API that group principal names are mapped to roles of the same name by default.

For more information about the Jakarta EE Security API in WebLogic Server, see *Developing Applications with the WebLogic Security Service*.

## Overview of the WebLogic Server Security Service

WebLogic Server includes a security architecture that provides a unique and secure foundation for applications that are available through the Internet. By taking advantage of the security features in WebLogic Server, enterprises benefit from a comprehensive, flexible security infrastructure designed to address the security challenges of making applications publicly available.

WebLogic security can be used standalone to secure WebLogic Server applications or as part of an enterprise-wide, security management system that represents a best-in-breed, security management solution. The key features of the WebLogic Security Service include:

- A comprehensive and standards-based design.
- End-to-end security for WebLogic Server-hosted applications, from the mainframe to the web browser.
- Legacy security schemes that integrate with WebLogic Server security, allowing companies to leverage existing investments.
- Security tools that are integrated into a flexible, unified system to ease security management across the enterprise.
- Easy customization of application security to business requirements through mapping of company business rules to security policies.
- A consistent model for applying security policies to Jakarta EE and application-defined resources.
- Easy updates to security policies. This release includes usability enhancements to the process of creating security policies as well as additional expressions that control access to WebLogic resources.
- Easy adaptability for customized security solutions.
- A modularized architecture, so that security infrastructures can change over time to meet the requirements of a particular company.
- Support for configuring multiple security providers, as part of a transition scheme or upgrade path.
- A separation between security details and application infrastructure, making security easier to deploy, manage, maintain, and modify as requirements change.
- Default WebLogic security providers that provide you with a working security scheme out of the box. This release supports additional authentication stores such as databases and gives the option to configure an external RDBMS system as a datastore to be used by select security providers.
- Customization of security schemes using custom security providers.
- Unified management of security rules, security policies, and security providers through the WebLogic Remote Console.
- Support for standard Jakarta EE security technologies such as the Java Authentication and Authorization Service (JAAS), Java Secure Sockets Extensions (JSSE), Java Cryptography Extensions (JCE), Java Authorization Contract for Containers (JACC), Java Authentication Service Provider Interface for Containers (JASPIC), and the Jakarta EE Security API (JSR 375).
- A foundation for web services security including support for Security Assertion Markup Language (SAML) 2.0.
- Capabilities which allow WebLogic Server to participate in single sign-on (SSO) with web sites, web applications, and desktop clients
- A framework for managing public keys which includes a certificate lookup, verification, validation, and revocation as well as a certificate registry.

# WebLogic Server Security Service Architecture

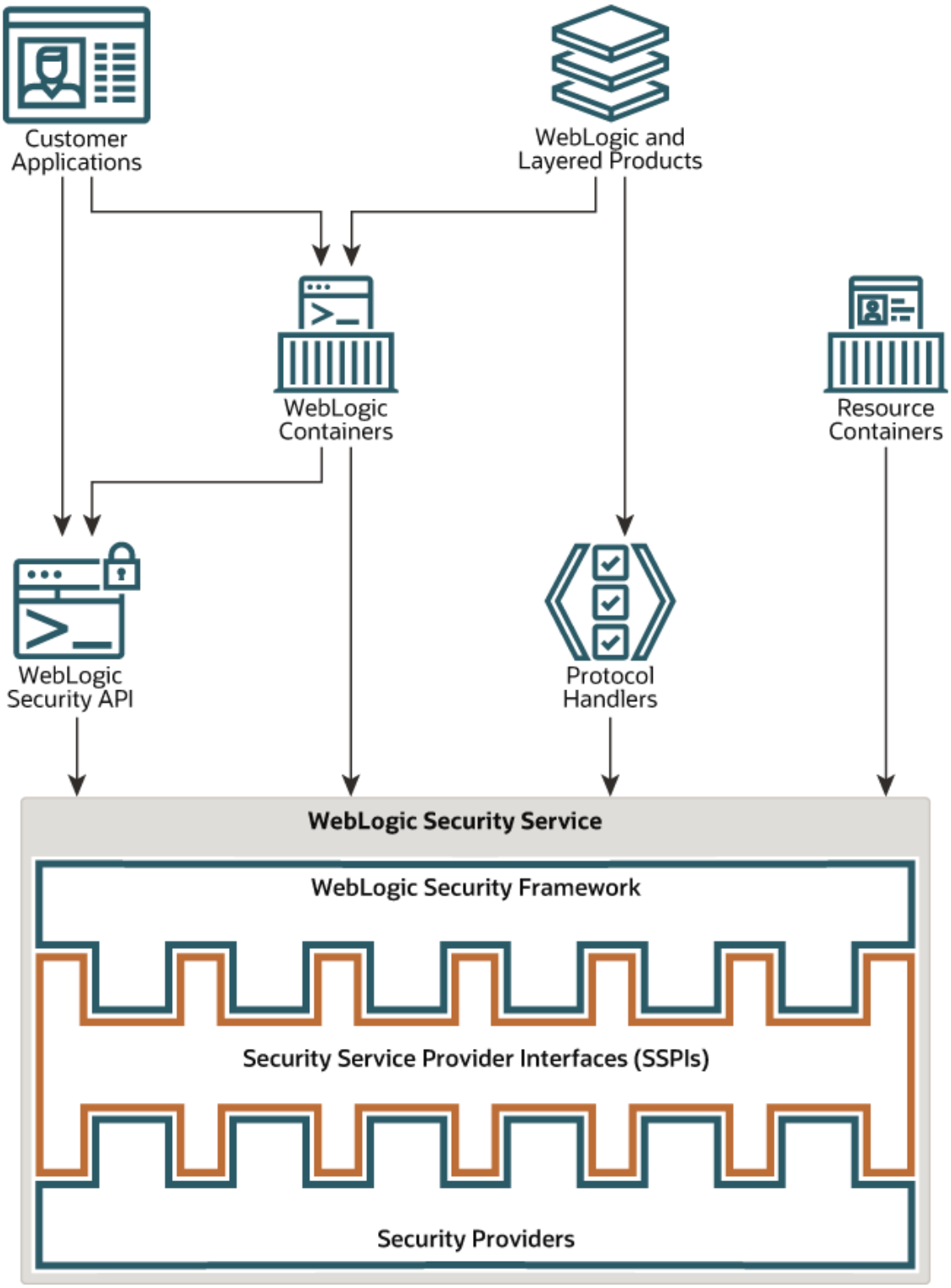
The WebLogic Server Security Service features a comprehensive and standards-based design that delivers end-to-end security for WebLogic Server-hosted applications from the mainframe to the web browser, easy customization of application security that can map of company business rules to security policies, a consistent model for applying security policies to Jakarta EE and application-defined resources, and more.

This section provides a description of the architecture of the WebLogic Security Service. The architecture encompasses the following major components:

## WebLogic Security Framework

Figure 10-1 shows a high-level view of the WebLogic Security Framework. The framework comprises interfaces, classes, and exceptions in the `weblogic.security.service` package.

**Figure 10-1 WebLogic Security Service Architecture**




The primary function of the WebLogic Security Framework is to provide a simplified application programming interface (API) that can be used by security and application developers to define security services. Within that context, the WebLogic Security Framework also acts as an

intermediary between the WebLogic containers (web and EJB), the resource containers, and the security providers.

## Single Sign-on with the WebLogic Server Security Framework

Single sign-on (SSO) is the ability to require a user to sign on to an application only once and gain access to many different application components, even though these components may have their own authentication schemes. SSO enables users to log in securely to all their applications, web sites, and mainframe sessions with just one identity. The Security Assertion Markup Language (SAML) and Windows Integrated Authentication features provide web-based SSO functionality for WebLogic Server applications.

## SAML Token Profile Support in WebLogic Web Services

The WebLogic web services and the WebLogic Security Framework support the generation, consumption, and validation of SAML 2.0 assertions. When using SAML assertions, a web service passes a SAML assertion and the accompanying proof material to the WebLogic Security Framework. If the SAML assertion is valid and trusted, the framework returns an authenticated Subject with a trusted principal back to the web service. WebLogic web services and the WebLogic Security Framework support the following SAML assertions:

- Sender-Vouches - The asserting party (different from the subject) vouches for the verification of the subject. The receiver must have a trust relationship with the asserting party.
- Holder-of-Key - The purpose of SAML token with "holder-of-key" subject confirmation is to allow the subject to use an X.509 certificate that may not be trusted by the receiver to protect the integrity of the request messages.

  Conceptually, the asserting party inserts an X.509 public certificate (or other key info) into a SAML assertion. (More correctly, the asserting party binds a key to a subject.) In order to protect this embedded certificate, the SAML assertion itself must be signed by the asserting entity. For WebLogic Server, the web service client signs the SAML assertion with its private key. That is, the signature on the assertion is the signature of the SAML authority, and is not based on the certificate contained in, or identified by, the assertion.
- Bearer - The subject of the assertion is the bearer of the assertion, subject to optional constraints on confirmation using attributes that may be included in the `<SubjectConfirmationData>` element of the assertion.

## The Security Service Provider Interfaces (SSPIs)

Security in WebLogic Server is based on a set of Security Service Provider Interfaces (SSPIs). The SSPIs can be used by developers and third-party vendors to develop security providers for the WebLogic Server environment. SSPIs are available for Adjudication, Auditing, Authentication, Authorization, Credential Mapping, Identity Assertion, Role Mapping, and Certificate Lookup and Validation.

The SSPIs allow customers to use custom security providers for securing WebLogic Server resources. Customers can use the SSPIs to develop custom security providers or they can purchase customer security providers from third-party vendors.

For more information on developing custom security providers, see *Developing Security Providers for Oracle WebLogic Server*.

## WebLogic Security Providers

Security providers are modules that "plug into" a WebLogic Server security realm to provide security services to applications. They call into the WebLogic Security Framework on behalf of applications.

If the security providers supplied with the WebLogic Server product do not fully meet your security requirements, you can supplement or replace them with custom security providers. You develop a custom security provider by:

- Implementing the appropriate security service provider interfaces (SSPIs) from the `weblogic.security.spi` package to create runtime classes for the security provider.
- Creating an MBean Definition File (MDF) and using the WebLogic MBeanMaker utility to generate an MBean type, which is used to configure and manage the security provider.

See *Developing Security Providers for Oracle WebLogic Server*.

# Managing WebLogic Server Security

When you manage WebLogic Server security, you focus primarily on two tasks: configuring security realms, and creating security policies.

### Security Realms

A security realm comprises mechanisms for protecting WebLogic resources. Each security realm consists of a set of configured security providers, users, groups, security roles, and security policies. A user must be defined in a security realm to be able to access any WebLogic resources defined in that realm. When a user attempts to access a WebLogic resource, WebLogic Server authenticates the user, and then authorizes access by determining whether the user can be mapped to the security role that is defined in the security realm and designated in the security policy for that WebLogic resource.

### Security Policies

A security policy answers the question, "Who has access to this WebLogic resource?" A security policy is an association between a WebLogic resource and one or more users, groups, or security roles that protects the WebLogic resource against unauthorized access. When creating a security policy, you can optionally define date and time constraints. A WebLogic resource has no protection until you assign it a security policy.

You assign security policies to any of the defined WebLogic resources (for example, an EJB resource or a JNDI resource) or to attributes or operations of a particular instance of a WebLogic resource (an EJB method or a servlet within a web application). If you assign a security policy to a type of WebLogic resource, all new instances of that resource inherit that security policy. Security policies assigned to individual resources or attributes override security policies assigned to a type of WebLogic resource.

# Security for Coherence

Three key security features are available for use when you deploy Oracle Coherence within an Oracle WebLogic Server domain: Oracle Coherence access controllers, Oracle WebLogic Server authorization, and Oracle Coherence identity tokens.

Coherence is secured using both WebLogic Server security components and Coherence-specific security components. The components include:

- SSL for authentication between Coherence cluster members
- SSL for authentication between extend clients (external to WebLogic Server) and a Coherence cluster
- WebLogic Server policies and roles for authorizing Coherence services and caches
- Identity assertion between extend clients and Coherence clusters

For details about configuring Coherence security, see Securing Oracle Coherence in Oracle WebLogic Server in *Securing Oracle Coherence*.

# Roadmap for Securing WebLogic Server

The WebLogic Server documentation set includes several introductory, procedural, and reference topics, including examples, that help you understand how to use the WebLogic Security Service.

**Table 10-1 Roadmap for Securing WebLogic Server**

| Major Task | Subtasks and Additional Information |
|---|---|
| Learning more about fundamental security concepts | • Auditing<br>• Authentication<br>• Security Assertion Markup Language (SAML)<br>• Single sign-on (SSO)<br>• Authorization<br>• Identity and trust<br>• Secure Sockets Layer (SSL)<br>• WebLogic security framework<br>• Single sign-on with the WebLogic Server security framework<br>• SAML token support in WebLogic web services<br>• Security Service Provider Interfaces (SSPIs)<br>• WebLogic security providers |
| Administering WebLogic Server security | • Security management concepts<br>• Customizing the default security configuration<br>• Migrating security data<br>• Managing the embedded LDAP server<br>• Managing the RDBMS security store<br>• Configuring keystores<br>• Configuring SSL<br>• Configuring cross-domain security<br>• *Securing Resources Using Roles and Policies for Oracle WebLogic Server*<br>• Exploring security options for cluster architectures<br>• Configuring Security for Coherence |

**Table 10-1 (Cont.) Roadmap for Securing WebLogic Server**

| Major Task | Subtasks and Additional Information |
|---|---|
| Authenticating users | • Authenticating users defined in an LDAP server<br>• Authenticating against an RDBMS system<br>• Authenticating a remote user<br>• Using SAML<br>• Configuring single sign-on<br>• Configuring single sign-on with Microsoft clients<br>• Configuring single sign-on with web browsers and HTTP clients<br>• Using Kerberos<br>• Using multiple authentication providers<br>• Configuring password composition rules<br>• Managing users and groups<br>• Using Java Authentication SPI for Containers (JASPIC)<br>• Using the Jakarta EE Security API |
| Configuring SSL | • Setting up SSL: main steps<br>• Configuring keystores<br>• Creating a keystore: example<br>• X.509 certificate revocation checking |
| Configuring authorization | • *Securing WebLogic Resources Using Roles and Policies*<br>• Configuring an authorization provider<br>• Using multiple authorization providers<br>• Using JAAS authorization<br>• Configuring a role mapping provider<br>• Using Java Authorization Contract Containers (JACC) |
| Learning more about security realms | • Introduction to security realms<br>• Users<br>• Groups<br>• Security roles<br>• Security policies<br>• Security providers |
| Programming applications for security | • *Developing Applications with the WebLogic Security Service*<br>• Configuring resource adapter security<br>• WebLogic web service security topics |
| Guidelines for locking down your WebLogic Server production environment | • *Securing a Production Environment for Oracle WebLogic Server* |

11

# WebLogic Server Web Services

A web service is a self-contained application that can be described, published, and invoked over a network, such as a corporate intranet or the Internet. Because you access web services using standard web protocols such as Extensible Markup Language (XML) and HTTP, the diverse and heterogeneous applications on the web (which typically already understand XML and HTTP) can access web services and communicate with each other automatically.

Major benefits of web services include:

- Interoperability among distributed applications that span diverse hardware and software platforms
- Easy, widespread access to applications through firewalls using web protocols
- A cross-platform, cross-language data model (XML) that facilitates developing heterogeneous distributed applications

For an overview of all web services supported in Oracle Fusion Middleware, see *Understanding Web Services*.

This chapter includes the following topics:

## Anatomy of a Web Service

Web services are characterized by three factors: what they do (the business functionality they expose), how they can be accessed (the set of published interfaces necessary to use the exposed functionality), and where they are (the web site which exposes that functionality).

What the web service can do is described in a standard XML vocabulary called Web Services Description Language (WSDL). For example, a banking web service may implement functions to check an account, print a statement, and deposit and withdraw funds. These functions are described in a WSDL file that any consumer can invoke to access the banking web service. As a result, a consumer does not have to know anything more about a web service than the WSDL file that describes what it can do.

A web service client (or consumer)--such as, a desktop application or a Jakarta Platform, Enterprise Edition portlet-- invokes a web service by submitting a request in the form of an XML document to the web service. The web service processes the request and returns the result to the web service client in an XML document.

The web service client can send a request in the form of a Simple Object Access Protocol (SOAP) message. SOAP is an XML messaging framework designed to allow heterogeneous applications to exchange structured information in a distributed environment. In turn, the web service processes the request and returns the response in a SOAP message.

You can also develop Representational State Transfer (REST) web services, or "RESTful" web services. REST describes any simple interface that transmits data over a standardized interface (such as HTTP) without an additional messaging layer, such as SOAP. REST provides a set of design rules for creating stateless services that are viewed as resources, or sources of specific information, and can be identified by their unique URIs. A client accesses the resource using the URI, a standardized fixed set of methods, and a representation of the resource is returned. The client is said to transfer state with each new resource representation.

To secure the message exchange, the web service may require credentials to access the service, for example a username and a password, or encrypt the response.

# Web Service Standards

Many specifications that define web service standards are written so as to allow for broad use of the specification throughout the industry. The implementation of a particular specification in Oracle WebLogic Server might not cover all possible usage scenarios covered by the specification. However, Oracle considers interoperability of web service platforms to be more important than providing support for all possible edge cases of the web service specifications.

Web services rely on a set of XML-based industry standards, including the following:

- XML, a data format that allows uniform communication between services consumers and services providers
- XML Schema, a framework that describes XML vocabularies used in business transactions
- SOAP, a protocol for exchanging structured information in the implementation of services
- WSDL, an XML-based language providing a model for describing services
- WS-Policy, a framework that provides a flexible and extensible grammar for describing the capabilities, requirements, and general characteristics of services using policies

# Roadmap for Web Services

The WebLogic Server documentation set includes several introductory, procedural, and reference topics, including examples, that help you understand how to develop and manage WebLogic web services.

**Table 11-1 Roadmap for Web Services**

| Major Task | Subtasks and Additional Information |
|---|---|
| Learning more about WebLogic web services | • Features and standards supported by WebLogic web services<br>• Overview of WebLogic web services<br>• Choose between Jakarta XML and RESTful web service<br>• Overview of web services security |
| Using the samples (for WebLogic web service developers) | • Sample application and code examples<br>• Jakarta XML Web Services example<br>• Examples of developing Jakarta XML Web Services clients<br>• JDeveloper web services tutorials |

**Table 11-1 (Cont.) Roadmap for Web Services**

| Major Task | Subtasks and Additional Information |
|---|---|
| Developing web services using Jakarta XML Web Services | • Starting from Java<br>• Starting from WSDL<br>• Programming the JWS file<br>• Using JAX binding<br>• Invoking a web service<br>• Invoking a web service asynchronously<br>• Using web services reliable messaging<br>• Managing web service persistence<br>• Configuring message buffering for web services<br>• Managing web services in a cluster<br>• Using web services atomic transactions<br>• Publishing a web service endpoint<br>• Using callbacks<br>• Optimizing binary data transmissions using MTOM/XOP<br>• Using XML catalogs<br>• Handling exceptions using SOAP<br>• Creating and using SOAP message handlers<br>• Programming stateful Jakarta XML Web Services using HTTP session |
| Developing RESTful web services | • Standards to use for RESTful web srevice development on WebLogic Server<br>• Learning about RESTful web service development<br>• Defining the root resource class<br>• Defining the relative URI of the root resource class<br>• Customizing request and response message types<br>• More advanced RESTful web service tasks<br>• Developing RESTful web service clients<br>• Packaging and Deploying RESTful web service applications<br>• Securing RESTful web services and clients<br>• Testing RESTful web services<br>• Monitoring RESTful web services<br>• Using server-sent events |
| Deploying and administering WebLogic web services | • Packaging and deploying RESTful web services<br>• Developing Jakarta XML Web Services |
| Securing WebLogic web services | • *Securing WebLogic Web Services for Oracle WebLogic Server* |
| Interoperability with WebLogic web services | • Interoperability with Microsoft WCF/.NET |

12

# Jakarta Enterprise Beans (EJBs)

Jakarta Enterprise Beans (EJBs) are Jakarta EE components that implement EJB technology. EJBs run in the EJB container, a runtime environment within Oracle WebLogic Server. Although transparent to the application developer, the EJB container provides system-level services, such as transactions and security, to the EJBs deployed in it. These services enable you to quickly build and deploy EJBs, which form the core of transactional Jakarta EE applications.

This chapter includes the following topics:

## Understanding EJBs

EJB 4.0 technology is the server-side component architecture for the development and deployment of component-based business applications. EJB technology enables rapid and simplified development of distributed, transactional, secure, and portable applications based on Jakarta EE technology.

The EJB 4.0 specification provides simplified programming and packaging model changes. The mandatory use of Jakarta interfaces from previous versions has been removed, allowing plain old Java objects to be annotated and used as EJB components. The simplification is further enhanced through the ability to place EJB modules directly inside of web applications, removing the need to produce archives to store the web and EJB components and combine them together in an EAR file.

This topic contains the following sections:

### EJB Documentation in WebLogic Server

See the following documents to know about using EJBs with WebLogic Server:

- For instructions on how to program, package, and deploy EJB 4.0 on WebLogic Server, see *Developing Enterprise JavaBeans for Oracle WebLogic Server*.
- For instructions on how to organize and build WebLogic Server EJBs in a split directory environment, see *Developing Applications for Oracle WebLogic Server*.
- For more information on programming and packaging 3.2 EJBs, see *Developing Enterprise JavaBeans, Using Deployment Descriptors*.

### Additional EJB Information

To learn more about EJB concepts, such as the benefits of enterprise beans, the types of enterprise beans, and their life cycles, then visit the following web sites:

- Jakarta Enterprise Beans 4.0 Specification at https://jakarta.ee/specifications/enterprise-beans/4.0/
- The "Enterprise Beans" section in the Java EE 8 Tutorial at https://javaee.github.io/tutorial/partentbeans.html#BNBLR

# Session EJBs Implement Business Logic

Session beans implement business logic. A session bean instance serves one client at a time. There are three types of session beans: stateful, stateless, and singleton.

For detailed information about the types of session beans and when to use them, see What Is a Session Bean in the Java EE 8 Tutorial.

### Stateful Session Beans

Stateful session beans maintain state information that reflects the interaction between the bean and a particular client across methods and transactions. A stateful session bean can manage interactions between a client and other enterprise beans, or manage a workflow.

**Example:** A company web site that allows employees to view and update personal profile information could use a stateful session bean to call a variety of other beans to provide the services required by a user, after the user clicks "View my Data" on a page:

- Accept the login data from a JSP, and invoke another EJB that validates the login data.
- Send confirmation of authorization to the JSP.
- Call a bean that accesses profile information for the authorized user.

### Stateless Session Beans

A stateless session bean does not store session or client state information between invocations—the only state it might contain is not specific to a client, for instance, a cached database connection or a reference to another EJB. At most, a stateless session bean may store state for the duration of a method invocation. When a method completes, state information is not retained.

Any instance of a stateless session bean can serve any client—any instance is equivalent. Stateless session beans can provide better performance than stateful session beans, because each stateless session bean instance can support multiple clients, albeit one at a time. The client of a stateless session bean can be a web service endpoint.

**Example:** An Internet application that allows visitors to click a "Contact Us" link and send an email could use a stateless session bean to generate the email, based on the "to" and "from" information gathered from the user by a JSP.

### Singleton Session Beans

Singleton session beans provide a formal programming construct that guarantees a session bean will be instantiated once per application in a particular Java Virtual Machine (JVM), and that it will exist for the life cycle of the application. With singletons, you can easily share state between multiple instances of an enterprise bean component or between multiple enterprise bean components in the application.

Singleton session beans offer similar functionality to stateless session beans but differ from them in that there is only one singleton session bean per application, as opposed to a pool of stateless session beans, any of which may respond to a client request. Like stateless session beans, singleton session beans can implement web service endpoints. Singleton session beans maintain their state between client invocations but are not required to maintain their state across server crashes or shutdowns.

**Example:** An Internet application that provides a central counter service. A stateless singleton bean can be called from a Jakarta client, with the count being consistently incremented by 1 as the client is invoked multiple times.

## Message-Driven Beans Implement Loosely Coupled Business Logic

A message-driven bean (MDB) implements loosely coupled or asynchronous business logic in which the response to a request need not be immediate. A message-driven bean receives messages from a JMS Queue or Topic, and performs business logic based on the message contents. It is an asynchronous interface between EJBs and JMS.

Throughout its life cycle, an MDB instance can process messages from multiple clients, although not simultaneously. It does not retain state for a specific client. All instances of a message-driven bean are equivalent—the EJB container can assign a message to any MDB instance. The container can pool these instances to allow streams of messages to be processed concurrently.

The EJB container interacts directly with a message-driven bean—creating bean instances and passing JMS messages to those instances as necessary. The container creates bean instances at deployment time, adding and removing instances during operation based on message traffic.

See *Developing Message-Driven Beans for Oracle WebLogic Server*.

**Example:** In an online shopping application, where the process of taking an order from a customer results in a process that issues a purchase order to a supplier, the supplier ordering process could be implemented by a message-driven bean. While taking the customer order always results in placing a supplier order, the steps are loosely coupled because it is not necessary to generate the supplier order before confirming the customer order. It is acceptable or beneficial for customer orders to "stack up" before the associated supplier orders are issued.

# EJB Anatomy and Environment

To develop an EJB, you must provide files for the EJB class, the business interfaces, and the helper classes. The environment for the EJB includes an EJB container and, optionally, one or more deployment descriptor files.

The following sections briefly describe classes required for each bean type, the EJB runtime environment, and the deployment descriptor files that govern a bean's runtime behavior:

## EJB Components

The composition of a bean varies by bean type. Table 12-1 defines the classes that make up each type of EJB, and defines the purpose of the class type.

**Table 12-1 EJB Components**

| EJB Component | Description | Stateless Session | Stateful Session | Singleton Session | MDB |
|---|---|---|---|---|---|
| Remote business interface | The remote business interface exposes business logic to remote clients—clients running in a separate application from the EJB. It defines the business methods a remote client can call. | Yes | Yes | Yes | No |

**Table 12-1 (Cont.) EJB Components**

| EJB Component | Description | Stateless Session | Stateful Session | Singleton Session | MDB |
|---|---|---|---|---|---|
| Local business interface | The local business interface exposes business logic to local clients—those running in the same application as the EJB. It defines the business methods a local client can call. | Yes | Yes | Yes | No |
| Local No-interface | The no-interface view is a variation of the Local view that exposes the public methods of the bean class without the use of a separate business interface. | Yes | Yes | Yes | Yes |
| Bean class | The bean class implements business logic. | Yes | Yes | Yes | Yes |

# The EJB Container

An EJB container is a runtime container for beans that are deployed to an application server. The container is automatically created when the application server starts up, and serves as an interface between a bean and runtime services such as:

- Life cycle management
- Code generation
- Security
- Transaction management
- Locking and concurrency control

# Embeddable EJB Container

Unlike traditional Jakarta EE server-based execution, embeddable usage allows client code and its corresponding enterprise beans to run within the same virtual machine and class loader. This provides better support for testing, offline processing (for example, batch jobs), and the use of the EJB programming model in desktop applications.

Most of the services present in the enterprise bean container in a Jakarta EE server are available in the embedded enterprise bean container, including injection, container-managed transactions, and security. Enterprise bean components execute similarly in both embedded and Jakarta EE environments, and therefore the same enterprise bean can be easily reused in both standalone and networked applications.

For more information about the Embedded Enterprise Bean Container, see Using an Embedded EJB Container in Oracle WebLogic Server in *Developing Enterprise JavaBeans for Oracle WebLogic Server*.

# EJB Metadata Annotations

The WebLogic Server EJB programming model uses the metadata annotations feature in which you create an annotated EJB bean file, and then use the WebLogic compile tool `weblogic.appc` (or its Ant equivalent `wlappc`) to compile the bean file into a Java class file and

generate the associated EJB artifacts, such as the required EJB interfaces and optional deployment descriptors.

See Programming the Annotated EJB Class in *Developing Enterprise JavaBeans for Oracle WebLogic Server*.

## Optional EJB Deployment Descriptors

As of EJB 3.0, you are no longer required to create the EJB deployment descriptor files (such as `ejb-jar.xml`). However, you can still use XML deployment descriptors if you want. In the case of conflicts, the deployment descriptor value overrides the annotation value.

If you are continuing to use deployment descriptors in your EJB implementation, refer to EJB Deployment Descriptors in *Developing Enterprise JavaBeans, Version 3.2, for Oracle WebLogic Server*.

WebLogic Server EJB has three deployment descriptors:

- `ejb-jar.xml`—The standard Jakarta EE deployment descriptor. All beans must be specified in an `ejb-jar.xml`. An `ejb-jar.xml` can specify multiple beans that will be deployed together.
- `weblogic-ejb-jar.xml`—WebLogic Server-specific deployment descriptor that contains elements related to WebLogic Server features such as clustering, caching, and transactions. This file is required if your beans take advantage of WebLogic Server-specific features. Similar to `ejb-jar.xml`, `weblogic-ejb-jar.xml` can specify multiple beans that will be deployed together.
- `weblogic-cmp-jar.xml`—WebLogic Server-specific deployment descriptor that contains elements related to container-managed persistence for entity beans. Entity beans that use container-managed persistence must be specified in a `weblogic-cmp-jar.xml` file.

For descriptions of the WebLogic Server EJB deployment descriptors, refer to Deployment Descriptor Schema and Document Type Definitions Reference in *Developing Enterprise JavaBeans, Version 3.2, for Oracle WebLogic Server*.

# EJBs Clients and Communications

An EJB can be accessed by server-side or client-side objects such as servlets, Jakarta client applications, other EJBs, web services, and non-Jakarta clients. Any client of an EJB, whether in the same or a different application, accesses it in a similar fashion. WebLogic Server automatically creates implementations of an EJB's home and business interfaces that can function remotely, unless the bean has only a local interface.
This topic includes the following sections:

## Accessing EJBs

Clients access enterprise beans either through a no-interface view or through a business interface. A no-interface view of an enterprise bean exposes the public methods of the enterprise bean implementation class to clients. Clients using the no-interface view of an enterprise bean may invoke any public methods in the enterprise bean implementation class or any superclasses of the implementation class. A business interface is a standard Java programming language interface that contains the business methods of the enterprise bean.

The client of an enterprise bean obtains a reference to an instance of an enterprise bean through either dependency injection, using Java programming language annotations, or JNDI

lookup, using the Java Naming and Directory Interface syntax to find the enterprise bean instance.

Dependency injection is the simplest way of obtaining an enterprise bean reference. Clients that run within a Jakarta EE server-managed environment, JavaServer Faces web applications, other enterprise beans, or Jakarta EE application clients, support dependency injection using the `jakarta.ejb.EJB` annotation.

Applications that run outside a Jakarta EE server-managed environment, such as Java SE applications, must perform an explicit lookup. JNDI supports a global syntax for identifying Jakarta EE components to simplify this explicit lookup. For more information see, Programming Access to EJB Clients in *Developing Enterprise JavaBeans for Oracle WebLogic Server*.

Because of network overhead, it is more efficient to access beans from a client on the same machine than from a remote client, and even more efficient if the client is in the same application.

## EJB Communications

WebLogic Server EJBs use:

- T3—To communicate with remote objects. T3 is a WebLogic-proprietary remote network protocol that implements the Remote Method Invocation (RMI) protocol.
- RMI—To communicate with remote objects. RMI enables an application to obtain a reference to an object located elsewhere in the network, and to invoke methods on that object as though it were co-located with the client on the same JVM locally in the client's virtual machine.

  An EJB with a remote interface is an RMI object. An EJB's remote interface extends `java.rmi.remote`. For more information on WebLogic RMI, see *Developing RMI Applications for Oracle WebLogic Server*.
- HTTP—An EJB can obtain an HTTP connection to a web server external to the WebLogic Server environment by using the `java.net.URL` resource connection factory. See Configuring EJBs to Send Requests to an URL in *Developing Enterprise JavaBeans, Version 3.2, for Oracle WebLogic Server*.

You can specify the attributes of the network connection an EJB uses by binding the EJB to a WebLogic Server custom network channel. See Configuring Network Resources in *Administering Server Environments for Oracle WebLogic Server*.

# Securing EJBs

By default, any user defined in the security realm can invoke the public methods of an EJB. Therefore, if you want to restrict access to EJBs, you can create security policies for them, or use security-related annotations, to specify the roles that are allowed to invoke all, or a subset, of its methods. For information about restricting access to EJBs using annotations within the EJBs themselves, see Securing Access to the EJB in *Developing Enterprise JavaBeans for Oracle WebLogic Server*.
In addition, you create security roles and map users to roles using the WebLogic Remote Console to update your security realm. For details, see Security Roles in *Oracle WebLogic Remote Console Online Help*.

For more information about security and EJBs:

- Security Fundamentals in *Understanding Security for Oracle WebLogic Server* has introductory information about authentication, authorization and other security topics.

- Securing Enterprise JavaBeans (EJBs) in *Developing Applications with the WebLogic Security Service* provides instructions on configuring authentication and authorization for EJBs.
- *Securing Resources Using Roles and Policies for Oracle WebLogic Server* contains instructions on configuring authentication and authorization for your EJBs using the WebLogic Remote Console.

# Roadmap for EJBs in WebLogic Server

The WebLogic Server documentation set includes several introductory, procedural, and reference topics, including examples, that help you understand how to develop, deploy, and secure EJBs in the WebLogic Server environment.

**Table 12-2 Roadmap for EJBs in WebLogic Server**

| Major Task | Subtasks and Additional Information |
|---|---|
| Understanding EJBs | • New Features and Changes in EJB<br>• What Is New and Changed in EJB 4.0 |
| Simple EJB examples | • Example of a simple stateless EJB<br>• Example of a simple stateful EJB<br>• Example of an interceptor class<br>• Packaged EJB 3.2 Examples in WebLogic Server<br>• Example of invoking an entity from a session bean |
| Iterative EJB developing | • Overview of the EJB development process<br>• Creating a source directory<br>• Programming access to EJB clients<br>• Programming and configuring transactions<br>• Programming the EJB interface<br>• Programming the EJB timer service<br>• Programming the annotated EJB class<br>• Programming optional interceptors<br>• Compiling Java source code<br>• Optionally creating and editing deployment descriptors<br>• Packaging EJBs<br>• Deploying EJBs |
| Programming the annotated EJB class | • Overview of metadata annotations and EJB bean files<br>• Programming the bean file: typical steps<br>• Complete list of metadata annotations by function |
| Deployment guidelines for EJBs | • Before you deploy an EJB<br>• Understanding and performing deployment tasks<br>• Deployment guidelines for EJBs |
| Using an embedded EJB container in Oracle WebLogic Server | • Overview of the embeddable EJB container |
| Configuring the Persistence Provider in Oracle WebLogic Server | • Overview of Oracle TopLink<br>• Specifying a persistence provider<br>• Using Oracle TopLink in Oracle WebLogic Server |
| Reference | • EJB metadata annotations reference |

13

# Monitoring, Diagnosing, and Troubleshooting

Oracle WebLogic Server provides several built-in management, diagnostic, and automation tools to increase management and operational efficiency of the server, the underlying JVM, deployed applications, and configured resources. These tools include the WebLogic Diagnostic Framework (WLDF), WebLogic logging services, integration with Oracle HotSpot, SNMP support, and more.

This chapter includes the following topics:

## WebLogic Diagnostics Framework

The WebLogic Diagnostics Framework (WLDF) is a monitoring and diagnostic framework that defines and implements a set of services that run within WebLogic Server processes and participate in the standard server life cycle. Using WLDF, you can create, collect, analyze, archive, and access diagnostic data generated by a running server and the applications deployed within its containers.

The data you collect using WLDF provides insight into the runtime performance of servers and applications and enables you to isolate and diagnose faults when they occur.

WLDF includes several components for collecting and analyzing data:

- WebLogic Server events can optionally be propagated to the Java Flight Recorder, a performance monitoring and profiling tool. WebLogic Server provides specific integration points with Java Flight Recorder:
  - WebLogic Server events are propagated to Java Flight Recorder for inclusion in a common data set for runtime or post-incident analysis.
  - The flight recording data is also included in WLDF diagnostic image captures, enabling you to capture flight recording snapshots based on WLDF watch rules. This full set of functionality enables you to capture and analyze runtime system information for both the JVM and the Fusion Middleware components running on it, in a single view.
- Diagnostic Image Capture—Creates a diagnostic snapshot from the server that can be used for post-failure analysis. The diagnostic image capture includes Java Flight Recorder data, if it is available, that can be viewed in Java Mission Control.
- Built-in Diagnostic Modules—Provides a simple and easy-to-use mechanism for performing basic health and performance monitoring of a WebLogic Server instance. The built-in diagnostic modules collect data from key WebLogic Server runtime MBeans that monitor the main components of a server instance, such as the WebLogic Server runtime, JDBC, JMS, and Jakarta EE containers hosting servlets and EJBs.
- Archive—Captures and persists data events, log records, and metrics from server instances and applications.
- Instrumentation—Adds diagnostic code to WebLogic Server instances and the applications running on them to execute diagnostic actions at specified locations in the code. The Instrumentation component provides the means for associating a diagnostic context with requests so they can be tracked as they flow through the system. The WebLogic Remote Console shows real-time and historical views of method performance information that has

been captured through the WLDF instrumentation capabilities, serving as a tool that can help identify performance problems in applications.

- Harvester—Captures metrics from runtime MBeans, including WebLogic Server MBeans and custom MBeans, which can be archived and later accessed for viewing historical data.
- Policies and Actions—Provides the means for monitoring server and application states and sending notifications based on criteria set in the watches.
- Monitoring Dashboard—The Monitoring Dashboard provides views and tools for graphically presenting diagnostic data about servers and applications running on them. The underlying functionality for generating, retrieving, and persisting diagnostic data is provided by the WebLogic Diagnostics Framework. The Monitoring Dashboard provides additional tools for presenting that data in charts and graphs.

  The diagnostic data displayed by the Monitoring Dashboard consists of runtime MBean attributes with numeric or Boolean values that are useful to measure, either as their current values or as their changes over time. These values, referred to in the Monitoring Dashboard as metrics, originate from one or more runtime MBean instances from one or more servers in the domain.

WLDF provides a set of standardized application programming interfaces (APIs) that enable dynamic access and control of diagnostic data, as well as improved monitoring that provides visibility into the server. Independent Software Vendors (ISVs) can use these APIs to develop custom monitoring and diagnostic tools for integration with WLDF.

WLDF enables dynamic access to server data through standard interfaces, and the volume of data accessed at any given time can be modified without shutting down and restarting the server.

# Logging Services

WebLogic logging services provide facilities for writing, viewing, filtering, and listening for log messages. These log messages are generated by WebLogic Server instances, subsystems, and Jakarta EE applications that run on WebLogic Server or in client JVMs.

WebLogic Server subsystems use logging services to provide information about events such as the deployment of new applications or the failure of one or more subsystems. A server instance uses them to communicate its status and respond to specific events. For example, you can use WebLogic logging services to report error conditions or listen for log messages from a specific subsystem.

By default, WebLogic logging services use an implementation based on the Logging APIs. In addition, WebLogic Server also provides the Server Logging Bridge, which provides a lightweight mechanism for applications that currently use Java Logging to have their log messages redirected to WebLogic logging services. Applications can use the Server Logging Bridge with their existing configuration; no code changes or programmatic use of the WebLogic Logging APIs is required.

# SNMP Support

WebLogic Server includes support for the Simple Network Management Protocol (SNMP), which you can use to enable enterprise-wide management systems for managing heterogeneous software and hardware environments from a single management console.

With SNMP, a manager sends a request for information about managed resources to an agent. The agent gathers the requested data and returns a response. You can also configure agents

to issue unsolicited reports (notifications) to managers when they detect predefined thresholds or conditions on a managed resource.

To request data about a specific managed resource, a manager must be able to uniquely identify the resource. In SNMP, each type of managed resource is described in a Management Information Base (MIB) as a managed object with a unique object identifier (OID). Individual organizations define their specific managed objects in MIB modules. Both manager and agent must have access to the same MIB module to communicate about specific managed resources.

# Custom JMX Applications

To integrate custom management systems with the WebLogic Server management system, WebLogic Server provides standards-based interfaces that are fully compliant with the Java Management Extensions (JMX) specification.

Software vendors can use these interfaces to monitor WebLogic Server MBeans, to change the configuration of a WebLogic Server domain, and to monitor the distribution (activation) of those changes to all server instances in the domain. While JMX clients can perform all WebLogic Server management functions without using Oracle's proprietary classes, Oracle recommends that remote JMX clients use WebLogic Server protocols (such as T3) to connect to WebLogic Server instances.

# Jakarta Management APIs

The Jakarta Management specification describes a standard data model for monitoring and managing the runtime state of any Jakarta EE web application server and its resources. It includes standard mappings of the model through a Jakarta Management EJB Component (MEJB).

The Jakarta Management APIs enable a software developer to create a single Java program that can discover and browse resources, such as JDBC connection pools and deployed applications, on any Jakarta EE web application server. The APIs are part of the Jakarta Management Specification, which requires all Jakarta EE web application servers to describe their resources in a standard data model.

# Roadmap for Monitoring, Diagnosing, and Troubleshooting in WebLogic Server

The WebLogic Server documentation set includes several introductory, procedural, and reference topics, including examples, that help you understand how to monitor, diagnose, and troubleshoot performance in WebLogic Server.

**Table 13-1 Roadmap for Monitoring, Diagnosing, and Troubleshooting in WebLogic Server**

| Major Task | Subtasks and Additional Information |
|---|---|
| Learning more about WLDF components | • Data creation, collection, and instrumentation<br>• Archive<br>• Policy and Action<br>• Data accessor<br>• Monitoring dashboard and request performance pages<br>• Diagnostic image capture<br>• Understanding WLDF configuration |
| Learning more about WebLogic logging services | • What you can do with WebLogic logging services<br>• Using message catalogs with WebLogic Server<br>• Logging components and environment<br>• Terminology<br>• Overview of the logging process<br>• Best practices for integrating Java logging with WebLogic logging services<br>• Server log files and domain log files<br>• Server and subsystem logs<br>• Log message format<br>• Message attributes<br>• Message severity<br>• Viewing WebLogic logging services<br>• Configuring WebLogic logging services<br>• Filtering WebLogic Server log messages<br>• Subscribing to messages<br>• Using the Server Logging Bridge |
| Using the Monitoring Dashboard | • About the monitoring dashboard interface<br>• Understanding how metrics are collected and presented<br>• The parts of a chart |
| Using SNMP with WebLogic Server | • WebLogic Server SNMP agents<br>• Security for SNMP<br>• MIB module for WebLogic Server<br>• Monitoring custom MBeans<br>• WebLogic Server notifications<br>• SNMP proxies<br>• WebLogic SNMP command-line utility |
| Creating JMX applications to manage WebLogic Server | • Developing custom management utilities with JMX<br>• Developing manageable applications with JMX<br>• Programming WebLogic deployment |
| Learning more about the Jakarta Management APIs | • JMO hierarchy<br>• JMO object names<br>• Optional features of JMOs<br>• Accessing JMOs<br>• Accessing the MEJB on WebLogic Server<br>• WebLogic Server extensions |

**Table 13-1 (Cont.) Roadmap for Monitoring, Diagnosing, and Troubleshooting in WebLogic Server**

| Major Task | Subtasks and Additional Information |
|---|---|
| Servers do not start | If you restart the server and encounter the following error, then follow the instructions in Doc ID 2691299.1 to solve the issue.<br><br>`<BEA-090402> <Authentication denied: Boot identity not valid; The user name and/or password from the boot identity file (boot.properties) is not valid.` |

14

# Sample Applications and Code Examples

WebLogic Server provides a rich set of code examples and sample applications that show several approaches to learning about and working with WebLogic Server. These examples and sample applications are available through a separate WLS examples installer.

The WebLogic Server code examples include a naming convention, which indicates the path names where the samples are located. Jakarta EE 8 Examples, Java EE 7 Examples, and Java EE 6 Examples refers to the names of the folders where these examples are installed and not the version of Java EE or Jakarta EE that the examples support. All the WebLogic Server examples remain relevant for developing WebLogic Server 15.1.1.0.0 applications.

This chapter includes the following topics:

## Installing and Running the Examples

You can create a complete WebLogic domain that is configured with the full set of deployed code examples and sample applications by using the WLS examples installation program.

This section contains the following topics:

### Installing the WebLogic Server Code Examples

You obtain the WebLogic Server and Coherence examples in a separate, examples JAR file, `fmw_15.1.1.0.0_wls_examples_generic.jar`, which you install in the **same** *`ORACLE_HOME`* as your WebLogic Server installation. For more information about obtaining and installing WebLogic Server, see Installing the Oracle WebLogic Server and Coherence Software.

To set up the WebLogic Server samples domains, you run the Configuration Wizard in Quick Start mode. Select to **Automatically Launch the Quickstart Configuration Wizard** on the last WLS Examples Installation screen. The Quick Start Wizard creates the three sample domains: `wl_server`, `medrec`, and `medrec-spring`, under *`ORACLE_HOME`*`\user_projects\domains`.

If you run the Quick Start Configuration Wizard *manually*, as described in the following steps, then you must create the sample domains one at a time. This process requires you to run the Quick Start Wizard three times to create all three domains. For more information, see Running the Quick Start Configuration Wizard in *Creating WebLogic Domains Using the Configuration Wizard*.

> **Note**
>
> During manual domain configuration, you provide the name for each sample domain, so it might be something other than `medrec`, `medrec-spring`, and `wl_server`.

- First, set the *`CONFIG_JVM_ARGS`* environment variable to specify the full path and JAR file name for each template that you want to use. For example, to create the MedRec domain:
  - On Windows:

```
set "CONFIG_JVM_ARGS=-DuserTemplates=C:\Oracle\Middleware\wlserver\common\
templates\wls\wls.jar,C:\Oracle\Middleware\wlserver\common\templates\
wls\medrec.jar"
```

- On UNIX:

```
export CONFIG_JVM_ARGS="-DuserTemplates=/Oracle/Middleware/wlserver/common/
templates/wls/wls.jar,/Oracle/Middleware/wlserver/common/templates/
wls/medrec.jar"
```

- Then, run the Configuration Wizard in Quick Start mode with the following command:
  - On Windows:

```
cd ORACLE_HOME\oracle_common\common\bin
config.cmd -target=config-oneclick
```

  - On UNIX:

```
cd ORACLE_HOME/oracle_common/common/bin
config.sh -target=config-oneclick
```

For detailed information, see Using Quick Start to Create the WebLogic Sample Domains in *Creating WebLogic Domains Using the Configuration Wizard*.

### MedRec-Spring

Before starting the MedRec-Spring domain, you have to copy the required Spring Framework 6.2.0 JAR files, which first must be downloaded from the https://repo1.maven.org/maven2/org/springframework repository. If needed, set your proxy settings for downloading the Spring Framework JAR files from the repository.

On Windows:

```
cd ORACLE_HOME\wlserver\samples\server
setExamplesEnv.cmd
cd medrec-spring
Edit project.properties to change the attribute “medrec-
spring.domain.dir=ORACLE_HOME\wlserver\samples\server\medrec-spring”
cd modules
ant -f module-build-commons.xml download.spring.pkgs // Download the Spring JAR files
copy "ORACLE_HOME\wlserver\samples\server\medrec-spring\lib\runtime\*.jar"
"ORACLE_HOME\wlserver\samples\server\medrec-
spring\modules\medrec\web\target\exploded\medrec\WEB-INF\lib\"  All
copy "ORACLE_HOME\wlserver\samples\server\medrec-spring\lib\runtime\*.jar"
"ORACLE_HOME\wlserver\samples\server\medrec-
spring\modules\physician\web\target\exploded\physician\WEB-INF\lib\"  All
cd ORACLE_HOME\user_projects\domains\medrec-spring
startWebLogic.cmd
```

On Linux/UNIX:

```
cd ORACLE_HOME/wlserver/samples/server/
sh setExamplesEnv.sh
cd  medrec-spring/modules/
ant -f module-build-commons.xml download.spring.pkgs // Download the Spring JAR files
cp -rf ORACLE_HOME/wlserver/samples/server/medrec-spring/lib/runtime/*.jar ORACLE_HOME/
wlserver/samples/server/medrec-spring/modules/physician/web/target/exploded/physician/
WEB-INF/lib/
cp -rf ORACLE_HOME/wlserver/samples/server/medrec-spring/lib/runtime/*.jar ORACLE_HOME/
wlserver/samples//server/medrec-spring/modules/medrec/web/target/exploded/medrec/WEB-
INF/lib
cd ORACLE_HOME/user_projects/domains/medrec-spring
sh startWebLogic.sh
```

## Starting the WebLogic Server Samples Domain

Start the examples server using one of the following procedures. In these procedures, *DOMAIN_HOME* represents the location where the samples domain is configured on your machine; for example, `C:\ORACLE_HOME\user_projects\domains`.

**On Windows**: Use a command shell and navigate to the *DOMAIN_HOME*`\<samples_domain>` directory. Enter the following command:

```
startWebLogic.cmd
```

**On UNIX Bourne Shell**: Navigate to the *DOMAIN_HOME*`/<samples_domain>` directory. Enter the following command:

```
sh ./startWebLogic.sh
```

> **Note**
>
> By default, the examples server uses port 7001 to listen for incoming connections. The MedRec server also uses the same listen port by default, which means that you cannot run both domains at the same time without changing one of the listen ports. If you want to run both domains at the same time, use the WebLogic Server Remote Console to change the listen port of the examples server to something other than 7001, and then restart it. You can then run the MedRec server using its default listen port at the same that you run the examples server.

## Running the WebLogic Server Code Examples

Review the instructions provided with the code examples for information about building, deploying and running the code examples. When you start the WebLogic Server examples domain, a browser is automatically launched that displays a web page from which you can browse the code examples and obtain instructions for building, deploying, and running them.

All the WLS code examples and sample applications run on the current release of WebLogic Server and support Jakarta EE 9.1.

# Conventions

The instructions for building, deploying, and running the WebLogic Server code examples include a number of typographical conventions for indicating the path names for entities such as the WebLogic Server installation directory, the samples domain home directory, Apache Derby, and so on, in a platform-neutral way.

The following conventions are used:

- The instructions generally are for Windows command shells. If you are using a UNIX or Linux-based shell, substitute `\` for `/` in path names.
- *ORACLE_HOME* represents the directory you specified as the Oracle Home when you installed WebLogic Server; for example, `C:\Oracle\Middleware\Oracle_Home`.
- *WL_HOME* represents the top-level installation directory for Oracle WebLogic Server. The default path is *ORACLE_HOME*`\wlserver`. (However, you are not required to install WebLogic Server in the Oracle Home directory.)

- *EXAMPLES_HOME* represents the directory in which the WebLogic Server code examples are configured. The default path is *ORACLE_HOME*\wlserver\samples\server.
- *DOMAIN_HOME* represents the directory in which the WebLogic Server sample domains are configured. The default path is *ORACLE_HOME*\user_projects\domains.

Source files for the code examples are separated from the domain configuration files, just as they should be in a real-world scenario. They are installed in the *EXAMPLES_HOME* directory.

The *DOMAIN_HOME*\<*samples_domain*> directory contains a WebLogic Server samples domain; it contains your applications and the XML configuration files that define how your applications and Oracle WebLogic Server will behave, as well as startup and environment scripts.

The *EXAMPLES_HOME*\examples\build directory contains client and server classes required by the examples and Derby database.

The *WL_HOME*\common\derby directory contains Derby, a demonstration database that the examples are configured to use. It also contains scripts that start and stop the database. For more information about Derby, see http://db.apache.org/derby.

# Jakarta EE 8 Examples

The Jakarta EE 8 examples demonstrate how to implement Jakarta EE 8 APIs and Oracle WebLogic Server-specific features. The Jakarta EE 8 examples are grouped into the following categories:

- Jakarta JSON Binding—Use Jakarta JSON Binding with JAX-RS.
- Jakarta Security—Configure a `DatabaseIdentityStore` to point to a backend database and then use it as an `IdentityStore`.
- Jakarta JSON Processing—Use JSON Patch, JSON Merge Patch, and JSON Pointer to update a JSON document.
- Servlet 4.0—Use the Servlet Mapping API, HTTP/2 Server Push, and the HTTP Trailer headers API.
- JAX-RS 2.1—Use the new Server-Sent Events (SSE) and the new REST Reactive Client API.
- JSF 2.3—Use the new features of JSF 2.3 such as direct support for WebSockets, class-level bean validation, CDI-compatible `@ManagedProperty` annotation feature, and Java date and time.
- JPA 2.2—Use injection in `@AttributeConverter` annotation, new Jakarta EE 8 `Date` and `Time`, and also use support for retrieving the results of `Query` and `TypedQuery` as streams.
- CDI 2.0—Use asynchronous events, observer ordering, and also use interception factory.

# Java EE 7 Examples

The Java EE 7 examples demonstrate how to implement Java EE 7 APIs and Oracle WebLogic Server-specific features. The Java EE 7 examples are grouped into the following categories:

- Batch Application Processing 1.0—Submit batch jobs and obtain information about submitted jobs using the JobOperator interface, and use the batch parallelization model to run partitioned job steps.
- Bean Validation 1.1—Use the bean validation group constraint and method level validation APIs.

- Context and Dependency Injection (CDI) 1.1—Use CDI events and the `@TransactionScoped` and `@Transactional` annotations.
- Concurrency Utilities 1.0—Create dynamic proxy objects using the `ContextService` interface, submit tasks using the `ManagedExecutorService` interface, submit delayed or periodic tasks using the `ManagedScheduledExecutorService` interface, and obtain a managed thread from the Java EE container using the `ManagedThreadFactory` interface.
- Enterprise JavaBeans 3.2—Use the new session bean life cycle callback interceptor methods API and also use a message-driven bean to implement a listener interface with no methods.
- Expression Language 3.0—Use new EL features, including support for a standalone environment, static field or method references, new operators, Lambda expressions, and collection constructions and operations.
- Java API for RESTful Web Services (JAX-RS) 2.0—Use asynchronous processing, filters and interceptors, and server-sent events (SSE) Jersey support.
- Java EE Connector Architecture 1.7—Develop a resource adapter and deploy connector resources with annotations defined in the Java EE Connector Architecture 1.7 specification.
- Java Message Service API 2.0—Use the JMS API in EJBs and servlets.
- Java Persistence 2.1—Use JPA criteria update and criteria delete APIs, and the stored procedures API.
- JSF 2.2—Use Java Server Faces (JSF) resource library contracts, file upload, faces flows, and HTML5 features.
- JSON Processing 1.0—Use the Java API for JSON processing with JAX-RS.
- Servlet 3.1—Use the HTTP protocol upgrade API, use non-blocking I/O for asynchronous reads and writes, change a session ID, and handle uncovered HTTP methods.
- WebSocket—Process JSON-format data, using CDI and EJBs in WebSocket endpoints, enable a server to echo text sent by a client, and enable fallback to HTTP long polling as an alternative for WebSocket messaging.

# Java EE 6 Examples

Oracle WebLogic Server includes code examples that demonstrate how to implement Java EE 6 APIs and WebLogic Server-specific features. The Java EE 6 examples are grouped in the following categories:

- Bean Validation 1.0: Use bean validation with JPA entities, JPA from Java SE, and JSF managed beans.
- Context and Dependency Injection (CDI) 1.0: Introduces CDI with type-safe dependency injection, interceptors, and producers.
- Data Source: Use the `@DataSourceDefinition` annotation.
- Enterprise JavaBeans 3.1: Use asynchronous methods, a calendar-based timer, simplified programming model and packaging in a WAR file, portable global JNDI names, and singleton session beans.
- Java API for RESTful Web Services (JAX-RS) 1.1: Build RESTful web services with JAX-RS.
- Java EE Connector Architecture 1.6: Use the Java EE Connector Architecture to connect two applications together using a stock trading application.

- JPA 2.0: Use the JPA Criteria Query API and the `@ElementCollection` mapping type.
- JSF 2.0: Incorporate Ajax in web applications, create bookmarkable web applications, and use facelets and templating.
- Servlet 3.0: Use annotations for servlets, filters, and listeners, handle file uploads with multipart files, and use asynchronous servlet and request handling, programmatic security, and servlet web fragments.

# Additional API Examples

Oracle WebLogic Server also includes a set of examples that demonstrate how to implement additional Java and Jakarta EE APIs and Oracle WebLogic Server-specific features. These examples are grouped in the following categories:

- Database Connectivity—Use Data Sources, Multi Data Sources, and Rowsets.
- EJB—Create stateless, stateful, entity, and message-driven EJBs, and more.
- Internationalization—Internationalize an application using simple message catalogs.
- Messaging—Use JMS topics, queues, and message-driven beans.
- Resource Adapter—Use an entity EJB to interact with a Jakarta Connector Architecture resource adapter.
- Security—Use the Java Authentication and Authorization Service, SAML, and outbound and two-way SSL.
- Transactions—Use JTA to perform distributed transactions using the two phase commit protocol across two XA resources.
- Web Application—Create simple servlets and JSPs, use the HTTP Publish-Subscribe server, and more.
- Web Services—Create a variety of web services using JWS annotations.
- XML—Use the STAX API and XMLBeans
- Cluster—Cluster an EJB and use HTTP session state replication.
- Coherence—Use the Coherence container to host Coherence applications
- WebLogic Scripting Tool—Use the WebLogic Scripting Tool (WLST) to configure and manage a running Administration Server instance.
- Split Development—Use the WebLogic split development directory structure to build, package, and deploy Enterprise Applications.
- Service Component Architecture—Use WebLogic SCA, a lightweight Spring 2.5 (or later) container, in a shopping cart application that demonstrates many of its key features.
- Spring—Use Spring-simplified configuration in a Spring-based web application.

# Avitek Medical Records

Avitek Medical Records (also known as MedRec) is a comprehensive educational sample application that demonstrates WebLogic Server and Jakarta EE features, as well as best practices. Avitek Medical Records is optionally installed with the WebLogic Server installation.

You can start MedRec from the *ORACLE_HOME*`/user_projects/domains/medrec` directory, where *ORACLE_HOME* is the directory you specified as the Oracle Home when you installed Oracle WebLogic Server.

The sample application, MedRec (Spring) demonstrates Spring Framework application development practices.

# Derby Open-Source Database

Derby is an open source relational database management system based on Java, JDBC, and SQL standards. Derby is bundled with WebLogic Server for use by the sample applications and code examples as a demonstration database.

For more information about Derby, see http://db.apache.org/derby.

15

# WebLogic Server Compatibility

Oracle attempts to support binary and source-level compatibility between Oracle WebLogic Server 15c (15.1.1.0.0) and versions 12.2.1.4.0, 14.1.1.0.0, and 14.1.2.0.0 in the areas of persistent data, generated classes, and API compatibility. In some cases, it is impossible to avoid incompatibilities. Where incompatibilities arise, they are fully documented in *Upgrading Oracle WebLogic Server*.
This chapter includes the following topics:

## Jakarta EE 9.1 Compatibility

WebLogic Server 15c (15.1.1.0.0) is Jakarta EE 9.1 compatible. This compatibility allows a Jakarta EE 9.1 compliant application to be developed on one operating system platform and deployed for production on another, without requiring Jakarta EE 9.1 application code changes.

Oracle ensures this compatibility of Jakarta EE 9.1 application portability within a WebLogic Server release level.

## Compatibility Within a Domain

Within the scope of a WebLogic domain, Oracle WebLogic Server supports a wide range of compatibility with respect to the specific versions of WebLogic Server instances that can run in that domain, as well as the mix of hardware, operating system, and JVM platforms on which those server instances can run.

However, depending upon the specific configurations present in the domain, such as WebLogic clusters, Oracle has specific recommendations for how you can achieve optimal performance. The following topics provide key information regarding compatibility within WebLogic domains:

### About WebLogic Server Version Numbers

Within a WebLogic domain, the Administration Server, Managed Server instances, and the domain itself each have a WebLogic Server version number. The version number contains five decimal places; for example, WebLogic Server 15.1.1.0.0. The meaning of each decimal place is described below:

- The first two decimal places together describe the Major Version number, for example "15.1" in **15.1**.1.0.0. The WebLogic Server 14.1 Major Version release is also branded as the WebLogic Server 14c Major Version release.
- The first three decimal places together describe the Minor Version number, for example "15.1.1" in **15.1.1**.0.0. WebLogic Server 15.1.1 (or 15.1.1.0.0) is the first Minor Version release of the WebLogic Server 15.1 Major Version release. WebLogic Server 15.1.2 (or 15.1.2.0.0) would be the second Minor Version release of the WebLogic Server 15.1 Major Version release.
- Patch Set releases for WebLogic Server 15.1.1.0.0 will increment the fourth decimal place. For example, 15.1.1.1.0 would be the first patch set release.
- Patch Set Update releases are named uniquely by incrementing the fifth decimal place with the date of the Patch Set Update release in *YYMMDD* format; for example,

15.1.1.0.251130. This convention is used for Patch Set Update naming purposes; for example, naming downloads available on My Oracle Support. However, the application of a Patch Set Update does not change the version number of an existing WebLogic Server installation as referenced in the Oracle inventory directory (`oraInventory`) used by WebLogic Server 15.1.1 installers.

You can obtain the version number and Patch Set level of a WebLogic Server instance or domain several different ways. For example:

- For an Administration Server or Managed Server instance, you can view the version message sent to `stdout` when the server is started. For example:

  ```
  <Version: WebLogic Server 15.1.1.0.0  Sat Nov 11 12:34:37 PDT 2025 1960751 >
  ```

- For a domain, you can view the value of the `<domain-version>` element in the domain configuration file, `config.xml`. For example:

  ```
  <domain-version>15.1.1.0.0</domain-version>
  ```

# WebLogic Version Compatibility

Within a WebLogic domain, the Administration Server, all Managed Server instances, and the WebLogic domain must be at the same WebLogic Server Major and Minor Version. This means that in WebLogic Server 15.1.1.0.0, the Administration Server, Managed Servers, and the WebLogic domain must all be at version 15.1.1.0.0. Note the following guidelines for maintaining consistency in Patch Set Update and Interim or One-off Patch levels within a domain.

- In general, the best practice is for all server instances within a domain to be at the same Patch Set Update (PSU) and Interim or One-off Patch level during steady-state operation. However, there may be cases where server instances are required to run at different PSUs and Interim or One-off Patch levels within a domain. The primary examples include:
  - When applying PSUs, Interim or One-off Patches in rolling fashion across server instances in the domain. In such cases, the maintenance should be applied to the Administration Server first, so that the Administration Server is at the same PSU and Interim or One-off Patch level (or higher) than its Managed Servers. See About Rolling Upgrade in *Upgrading Oracle WebLogic Server*.
  - When there are specific requirements to run Managed Servers within a domain at different PSU and Interim or One-off Patch levels in steady-state operation. In such cases, the Administration Server should be at the highest PSU level, so that the Administration Server is at the same PSU level or higher than all of the Managed Servers. If Managed Servers within a domain are running with different Interim or One-off Patches, it will not be possible to apply a consistent set of Interim or One-off Patches to the Administration Server. Because this maintenance complexity may be difficult to manage, the general best practice is to use the same PSU and Interim or One-off Patch level across all servers in the domain.
- Server instances within a cluster or domain can run on any hardware and operating systems as long as the hardware and operating systems are listed on the Oracle Fusion Middleware Supported System Configurations page on Oracle Technology Network. However, note that running clustered Managed Server instances on different hardware and operating systems may impact load balancing and performance. In general, the best practice is to run all Managed Servers within a cluster on the same hardware and operating system.
- If the WebLogic domain is part of an Oracle Enterprise Manager Cloud Control installation, additional requirements exist regarding the combinations of hardware, operating system,

and JVMs, that may be configured in the domain. See *Oracle Enterprise Manager Cloud Control Administrator's Guide*.

For more information about WebLogic domains and additional details about domain compatibility, see Domain Restrictions in *Understanding Domain Configuration for Oracle WebLogic Server*.

## Hardware, Operating System, and JVM Platform Compatibility

WebLogic Server instances within a domain can run on any hardware, operating system, and JVM platform as long as the hardware, operating systems, and JVMs are supported for the current version of WebLogic Server. For details, see the Oracle Fusion Middleware Supported System Configurations page on the Oracle Technology Network.

> **Note**
>
> Although this platform compatibility support extends to Managed Server instances within a cluster, Oracle strongly recommends that clusters be homogeneous with respect to the underlying hardware, operating system, and JVM. Managed Server instances running in the same cluster are assumed to be equivalent, so running clustered server instances on mixed platforms may have a negative impact on load balancing and performance. If you must operate a cluster on a mixed platform, Oracle strongly recommends that you understand the load balancing and performance implications.

## Node Manager Compatibility

As a best practice, Oracle recommends that the version of Node Manager used in a WebLogic domain should match the version of the Administration Server.

# Persistent Data Compatibility

If you are upgrading Oracle WebLogic Server from release 12.2.1.4.0, 14.1.1.0.0, or 14.1.2.0.0 to release 15c (15.1.1.0.0), a number of updates are required to several configuration files. However, Oracle WebLogic Server upgrade tooling makes those configuration file updates for you automatically.

# API Compatibility

WebLogic Server 12.2.1.4.0, 14.1.1.0.0, and 14.1.2.0.0 applications deployed on WebLogic Server must be modified and recompiled to run in the WebLogic Server 15.1.1.0.0 application environment.
Oracle recommends using Rewrite WebLogic recipes to apply the changes required for migrating applications to WebLogic Server 15.1.1.0.0 and Jakarta EE 9.1. You can use Rewrite WebLogic recipes for migrating WebLogic Server applications to newer versions of WebLogic Server, Java, Jakarta EE, and related versions of Jakarta Server Faces and Spring Framework. For more information, see Upgrade Your Applications in *Upgrading Oracle WebLogic Server*.

For a list of previously deprecated APIs that are removed in Oracle WebLogic Server 15.1.1.0.0, see Removed Functionality and Components in *What's New in Oracle WebLogic Server 15.1.1.0.0*.

# Protocol Compatibility

Interoperability between WebLogic Server 15c (15.1.1.0.0) and WebLogic Server 12.2.1.4.0, 4.1.1.0.0, and 14.1.2.0.0 is supported in several scenarios with regard to WebLogic clients, transport protocols, and WebLogic proxy plug-ins.

The supported client-server interoperability scenario is as follows: a WebLogic Server 15.1.1.0.0 server acting as a client can invoke RMI-based applications hosted on a WebLogic Server 14.1.2.0.0 server using IIOP*, T3, T3S, HTTP, and HTTPS. JMS applications can be invoked using T3, T3S, HTTP, and HTTPS.

The supported server-server interoperability scenarios are as follows:

- WLS 15.1.1.0.0 <-> WLS 14.1.2.0.0
- WLS 15.1.1.0.0 <-> WLS 14.1.1.0.0
- WLS 15.1.1.0.0 <-> WLS 12.2.1.4.0

The following clients can interoperate with WebLogic 15.1.1.0.0 servers. The parentheses indicate which version of WLS provides the client in its installation and whether the client itself uses `javax` or `jakarta` packages.

- WL Thin T3 Client (15.1.1.0.0 `jakarta`)
- WL Thin T3 Client (14.1.2.0.0 `javax`)
- WL Thin T3 Client (14.1.1.0.0 `javax`)
- WL Thin T3 Client (12.2.1.4.0 `javax`)
- Install Client (T3) (15.1.1.0.0 `jakarta`)
- Install Client (T3) (14.1.2.0.0 `javax`)
- Install Client (T3) (14.1.1.0.0 `javax`)
- Install Client (T3) (12.2.1.4.0 `javax`)
- Web Services (15.1.1.0.0 `jakarta`)
- Web Services (14.1.2.0.0 `javax`)
- Web Services (14.1.1.0.0 `javax`)
- Web Services (12.2.1.4.0 `javax`)
- RESTful Webservices Client (15.1.1.0.0 `jakarta`)
- RESTful Webservices Client (14.1.2.0.0 `javax`)
- RESTful Webservices Client (14.1.1.0.0 `javax`)
- RESTful Webservices Client (12.2.1.4.0 `javax`)

WebLogic Server 15c (15.1.1.0.0) supports client to server IIOP communications and interoperability with the following restrictions:

- The only supported WebLogic Java IIOP clients are:
  - A `weblogic.jar` install client from 15.1.1.0.0 or earlier WebLogic Server versions that are supported and under error correction.
  - A `wlfullclient.jar`, available in earlier versions of WebLogic Server that are supported and under error correction. The `wlfullclient.jar` is included but

deprecated in 12.2.1.4.0. WebLogic Server versions 14.1.1.0.0 and 14.1.2.0.0 do not include `wlfullclient.jar`.

  - A `wlclient.jar` from earlier WebLogic Server versions that are supported and under error correction. WebLogic Server 15c (15.1.1.0.0) does not include `wlclient.jar`.

- A Jakarta SE client (no JAR file from WebLogic Server in the class path) is *not* supported.
- For WebLogic Server 15c (15.1.1.0.0) instances running on JDK17 or 21, IIOP interoperability with Jakarta clients is only available with a WebLogic Server 15c (15.1.1.0.0) install client running on JDK 17 or 21.
- Interoperability support is *not* available between WebLogic Server 15c (15.1.1.0.0) instances running on JDK 17 or 21, and WebLogic Java IIOP clients running on JDK 17 or 21.

> **Note**
>
> Oracle recommends using a T3 capable WebLogic Java client and the T3 protocol instead of IIOP when possible. It is rare that the IIOP protocol is required for Java to Java communications; T3 is more efficient than IIOP. For more information, see Overview of Standalone Clients in *Developing Standalone Clients for Oracle WebLogic Server*.