



Integrated Cloud Applications & Platform Services



Oracle WebLogic Server 12c: Administration II

Student Guide – Volume I
D80153GC20
Edition 2.0 | December 2016 | D96006

Learn more from Oracle University at education.oracle.com

Authors

Mark Lindros
TJ Palazzolo
Al Saganich
Tom Eliason

Technical Contributors and Reviewers

Bill Bell
Elio Bonazzi
Tom McGinn
Eduardo Moranchel Rosales
Will Lyons
David Cabelus
Greg Stachnick
Donna Micozzi
Jon Patt
Matthew Slingsby
Bill Albert
Rich Whalen
Kevin Tate
Serge Moiseev
Takyiu Liu
Angelika Krupp
Viktor Tchemodanov
Diganta Choudhury
Jose Alvarez
Alexander Ryndin

Editor

Raj Kumar

Graphic Designers

Seema Bopaiah
Kavya Bellur

Publishers

Sujatha Nagendra
Raghunath M
Asief Baig

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

1 Course Introduction

- Course Objectives 1-2
- Target Audience 1-4
- Course Prerequisites 1-5
- Introductions and Setting Expectations 1-6
- Course Schedule 1-7
- Course Practices 1-10
- Classroom Guidelines 1-11
- Facilities in Your Location 1-12
- For More Information 1-13
- Related Training 1-14

2 WebLogic Server Review

- Objectives 2-2
- Agenda 2-3
- Oracle WebLogic Server: Overview 2-4
- WebLogic Server Domain 2-6
- Administration Server 2-7
- Managed Servers 2-8
- Machines and Clusters 2-9
- WebLogic Server Application Services 2-10
- WebLogic Server Application: Example 2-11
- Agenda 2-12
- Start Scripts 2-13
- Node Manager 2-14
- WebLogic Tools: Administration Console 2-15
- WebLogic Tools: WebLogic Scripting Tool (WLST) 2-16
- Deployment 2-17
- Summary 2-18

3 Upgrading WebLogic Server

- Objectives 3-2
- Agenda 3-3
- Definition of Upgrade 3-4
- Patch 3-5

Minor Upgrade	3-6
Major Upgrade	3-7
Quiz	3-8
Agenda	3-9
What Is a Rolling Upgrade?	3-10
Multiple Installation and Domain Locations	3-11
Leverage WebLogic Clusters to Avoid Down Time	3-12
Quiz	3-14
Agenda	3-15
Rolling Upgrade Process: Overview	3-16
Backup	3-17
Shutdown	3-18
Upgrade	3-19
Restart	3-20
Quiz	3-21
Summary	3-22
Practice 3-1 Overview: Setting Up the Course Practice Environment	3-23
Practice 3-2 Overview: Performing a Rolling Upgrade	3-24

4 Creating and Using Domain Templates

Objectives	4-2
Agenda	4-3
Domain Template Review	4-4
Agenda	4-5
Template Contents	4-6
Template Exclusions	4-7
Script Replacement Variables	4-8
Agenda	4-9
Extension Template: Concepts	4-10
Agenda	4-11
Fusion Middleware (FMW) Templates	4-12
Quiz	4-13
Agenda	4-15
Why Use Custom Templates	4-16
Template Builder	4-17
Creating a Domain Template	4-18
Selecting the Template Domain Source	4-19
Configuring Applications in a Template	4-20
Adding Files to the Template	4-21
Preparing Scripts and Files with Variables	4-22
Review Configuration and Create Template	4-23

SQL Scripts and LDAP Data	4-24
Creating an Extension Template	4-26
Using a Custom Template with the Configuration Wizard	4-28
Post Domain Creation Tasks	4-34
Using a Custom Extension Template with the Configuration Wizard	4-35
Summary	4-37
Practice 4-1 Overview: Creating and Using a Custom Domain Template	4-38

5 WebLogic Server Startup and Crash Recovery

Objectives	5-2
Agenda	5-3
Node Manager Architecture	5-4
Node Manager Default Behavior	5-5
Configure Java-Based Node Manager	5-6
Agenda	5-7
Starting the Node Manager at System Startup	5-8
Configuring the Node Manager as a Windows Service	5-9
Configuring the Node Manager on UNIX Systems	5-10
Configuring the Node Manager as an inetd Service	5-11
Registering and Managing inetd Services	5-14
Configuring the Node Manager as an xinetd Service	5-15
Solaris	5-17
Example SMF Manifest File	5-18
Set Up and Start Node Manager with SMF	5-21
Agenda	5-22
How the Node Manager Restarts an Administration Server	5-23
How the Node Manager Restarts a Managed Server	5-24
RestartInterval and RestartMax	5-25
Crash Recovery	5-26
CrashRecoveryEnabled	5-27
Quiz	5-28
Summary	5-29
Practice 5-1 Overview: Configuring Automatic Start and Restart of a System	5-30

6 WebLogic Scripting Tool (WLST)

Objectives	6-2
Agenda	6-3
WebLogic Scripting Tool (WLST)	6-4
Agenda	6-5
Jython	6-6
Using Jython	6-7

Variable Declaration	6-8
Conditional Expressions	6-9
Loop Expressions	6-10
I/O Commands	6-11
Exception Handling	6-12
Quiz	6-13
Agenda	6-14
WLST Modes	6-15
WLST Example	6-16
Running WLST Scripts	6-17
WLST Development Tools	6-18
configToScript()	6-19
Script Recording Using the Administration Console	6-20
Oracle Enterprise Pack for Eclipse	6-22
WLST Command Tips	6-23
General WLST Commands	6-24
Offline WLST Commands	6-25
Online WLST Commands	6-26
Quiz	6-27
Agenda	6-28
WebLogic JMX Overview	6-29
Configuration MBeans	6-30
Runtime MBeans	6-31
WebLogic Server MBean Examples	6-32
MBean Properties in the Administration Console	6-33
Browsing MBean Documentation	6-34
Referencing MBeans in WLST	6-36
Quiz	6-37
Agenda	6-38
Creating a Template and a Domain	6-39
Connecting to a Server	6-40
Password Management	6-41
WLST Variables	6-42
Password Management	6-43
Configuring a Server	6-44
Monitoring a Server	6-45
Adding a Server to a Cluster	6-46
Creating a Data Source	6-47
Dynamic Cluster Scaling	6-49
Monitoring a Data Source	6-50
Creating an LDAP Authentication Provider	6-51

Modifying a Domain Offline	6-52
Deploying an Application	6-53
Creating a Dynamic Cluster	6-54
Scaling Up a Dynamic Cluster	6-56
Scaling Down a Dynamic Cluster	6-57
Quiz	6-58
Agenda	6-59
Some FMW Commands	6-60
Summary	6-61
Practice 6-1 Overview: Creating and Modifying a Domain with WLST	6-62
Practice 6-2 Overview: Monitoring a Domain with WLST	6-63

7 RESTFul Management Services

Objectives	7-2
Agenda	7-3
What Is REST?	7-4
WebLogic Server and REST	7-5
What Can Be Monitored Using WebLogic REST?	7-6
Clients and WebLogic REST	7-7
Enabling REST Management	7-8
Multitenant (MT) Versus Nonmultitenant environments	7-9
Supported Representations	7-10
REST Resource Access Roles	7-11
Agenda	7-12
WebLogic RESTful Interfaces	7-13
REST Operations Flow	7-14
Example: Returning Server Information Using cURL	7-15
Typical REST Operations	7-16
Common WebLogic REST Results	7-17
WebLogic REST Error Handling	7-19
Understanding WebLogic REST Requests	7-20
WebLogic REST MBean Hierarchy	7-21
WebLogic Legacy REST Resources	7-22
WebLogic REST Response Bodies	7-23
Example: Listing Server Information by Name	7-24
Invoking Operations	7-25
Create Forms	7-26
Creating Resources	7-28
Agenda	7-29
Common Workflows Modify a configuration using an edit	7-30
Common Workflows Deploy/View/Un deploy an application	7-31

Common Workflows Start/View/Stop Servers	7-32
Common Workflows Monitor Servers	7-33
Summary	7-34
Quiz	7-35
Practice 7-1 Overview: Creating and Managing Servers Using REST	7-36
WebLogic REST Monitoring Relative Paths	7-37

8 Secure Sockets Layer (SSL)

Objectives	8-2
Agenda	8-3
What Is SSL?	8-4
SSL Terminology	8-5
Symmetric Encryption and Decryption	8-6
Asymmetric Encryption and Decryption	8-7
Digital Certificates	8-9
Digital Certificate: Example	8-10
Certificate Authorities	8-11
SSL Communication	8-12
One-Way SSL Handshake	8-13
Two-Way SSL Handshake	8-14
Quiz	8-15
Agenda	8-17
WebLogic and SSL	8-18
Demo Certificates	8-19
WebLogic SSL Requirements	8-20
WebLogic SSL Connections	8-21
Agenda	8-23
What Is a Keystore?	8-24
keytool Utility	8-25
Working with a Keystore	8-26
Configuring WebLogic Keystores	8-28
Quiz	8-29
Agenda	8-30
Configuring SSL for WebLogic	8-31
SSL and WebLogic Proxy Plug-Ins	8-32
SSL and Oracle HTTP Server	8-34
SSL and Hardware Load Balancers	8-35
Quiz	8-36
Summary	8-37
Practice 8-1 Overview: Setting Up SSL	8-38

9 Shared Java EE Libraries

- Objectives 9-2
- Agenda 9-3
- Before Shared Libraries 9-4
- Shared Libraries 9-5
- Types of Shared Libraries 9-6
- Java EE Library Support 9-7
- WebLogic Java EE Shared Libraries 9-8
- Configuration Precedence and Deployment 9-9
- Managing Precedence 9-10
- Configuring a Shared Library 9-11
- Configuring a Shared Library in the Manifest File 9-12
- Referencing a Shared Library 9-13
- Quiz 9-15
- Agenda 9-16
- Deploying a Library with the Console 9-17
- Deploying a Library with weblogic.Deployer 9-18
- Deploying a Library with WLST 9-19
- Summary 9-20
- Practice 9-1 Overview: Configuring and Deploying a Shared Library 9-21

10 Application Work Managers

- Objectives 10-2
- Agenda 10-3
- WebLogic Server Threads 10-4
- Monitoring a Server Thread Pool 10-5
- Monitoring Server Threads 10-6
- Stuck Thread Handling 10-7
- Configuring Stuck Thread Handling 10-8
- Application Stuck Thread Handling 10-9
- Quiz 10-10
- Agenda 10-11
- Work Managers 10-12
- Work Manager Scope 10-13
- Work Manager Architecture 10-14
- Quiz 10-15
- Agenda 10-16
- Request Classes 10-17
- Creating a Work Manager 10-18
- Creating a Request Class 10-19
- Constraints 10-20

Creating a Constraint	10-21
Work Manager WLST Example	10-22
Work Managers and Stuck Threads	10-23
Assigning Work Managers to Applications	10-24
Quiz	10-25
Summary	10-26
Practice 10-1 Overview: Creating and Using Work Managers	10-27

11 Working with the Security Realm

Objectives	11-2
Agenda	11-3
Security Realm: Review	11-4
Default Security Configuration	11-6
Agenda	11-7
How WebLogic Resources Are Protected	11-8
Examples of WebLogic Resources to Protect	11-9
Users and Groups	11-10
Group Membership	11-11
Roles	11-12
Policies	11-13
Configuring New Users	11-14
Configuring New Groups	11-15
Configuring Group Memberships	11-16
Configuring New Roles	11-17
What Is Role Mapping?	11-18
Configuring Role Mapping	11-19
Configuring Roles Using WLST	11-20
Configuring New Policies	11-21
Configuring Policies Using WLST	11-22
Security Configuration Sources	11-23
Configuring Sources Using WLST and weblogic.Deployer	11-24
Deployment Descriptor Security Example: weblogic.xml	11-25
Deployment Descriptor Security Example: web.xml	11-26
Embedded LDAP Server	11-27
Configuring the Embedded LDAP Server	11-28
Quiz	11-30
Practice 11-1 Overview: Creating Users, Groups, Roles, and Policies	11-31
Agenda	11-32
Auditing	11-33
Sample Auditing Output	11-34
Security Audit Events	11-35

WebLogic Auditing Architecture	11-36
Custom Versus Default Auditing Provider	11-37
Creating the Default Auditing Provider	11-38
Configuring the Default Auditing Provider	11-39
Configuration Auditing	11-41
Quiz	11-43
Summary	11-44
Practice 11-2 Overview: Configuring WebLogic Auditing	11-45

12 Disaster Recovery and Migration

Objectives	12-2
Agenda	12-3
Disaster Recovery Concepts	12-4
Site Symmetry	12-6
Recommended Architecture	12-7
General Best Practices	12-8
Hosts File: Example	12-9
Quiz	12-10
Agenda	12-11
Administration Server Review	12-12
Impact of Administration Server Failure	12-13
Backing Up a Domain Configuration	12-14
Recovery of the Administration Server Configuration	12-15
Restarting an Administration Server on a New Computer	12-16
Quiz	12-17
Agenda	12-18
Java Transaction API (JTA) Review	12-19
What Is Service Migration?	12-20
Service Migration Prerequisites	12-21
Service Migration Architecture: Database Leasing	12-22
Service Migration Architecture: Consensus Leasing	12-23
What Is a Migratable Target?	12-24
Service Migration Policy Options	12-25
Configuration Roadmap	12-26
JTA Service Migration: Before Failure	12-27
JTA Service Migration: After Failure	12-28
Configuring JTA Service Migration	12-29
Set Up Automatic JTA Service-Level Migration	12-30
Quiz	12-31
Practice 12-1 Overview: Configuring JTA Service-Level Migration	12-34
Agenda	12-35

Whole-Server Migration 12-36
Automatic Server Migration Architecture: No Failure 12-37
Automatic Server Migration Architecture: Machine Failure 12-38
Configuration: Overview 12-39
Quiz 12-40
Summary 12-41

13 Domain Partitions

Objectives 13-2
Agenda 13-3
Multitenancy 13-4
Hardware Isolation 13-5
Network Isolation 13-6
Application Isolation 13-8
Security Isolation 13-9
Agenda 13-10
What Is WebLogic Server 12c Multitenancy? 13-11
Understanding Tenants and Partitions 13-12
What Is a Domain Partition? 13-13
WebLogic Server Multitenancy End-to-End Integration 13-14
WebLogic Server Multitenancy Partition Isolation 13-15
WebLogic Server Multitenancy Components 13-16
What Are Resource Groups? 13-17
What Are Resource Group Templates? 13-18
Resource Group Templates Simplify Management 13-19
Virtual Targets 13-20
Virtual Target Versus Virtual Host 13-23
Oracle Traffic Director 12.2.1 13-24
Partition Security 13-25
Partition Work Managers 13-26
Multitenancy Examples 13-28
Single Partition Use Case 13-29
Agenda 13-30
Creating Virtual Targets 13-31
Creating Domain Partitions 13-33
Working with Domain Partitions 13-36
Importing and Exporting Domain Partitions 13-37
Completing Domain Partition Import (or export) 13-39
Application Deployment 13-41
Quiz 13-42
Summary 13-43
Practice 13-1 Overview: Creating and Working with Domain Partitions 13-44

14 Managing Data Sources

- Objectives 14-2
- Agenda 14-3
- Review: JDBC 14-4
- Data Source: Review 14-5
- XA Data Source Review 14-6
- Agenda 14-7
- Why Manage a Data Source? 14-8
- Suspend a Data Source 14-9
- Resume a Data Source 14-10
- Practice 14-1 Overview: Controlling a Data Source 14-11
- Agenda 14-12
- Oracle Real Application Clusters (RAC): Overview 14-13
- Oracle GridLink for RAC 14-14
- Agenda 14-15
- Multi Data Sources 14-16
- Multi Data Source Architecture 14-17
- Comparison of GridLink and Multi Data Sources 14-18
- Failover Option 14-19
- Load Balancing Option 14-20
- Quiz 14-21
- Agenda 14-22
- Connection Testing 14-23
- Agenda 14-25
- Proxy Data Sources 14-26
- Creating a Proxy Data Source 14-27
- Monitoring Proxy Data Source JDBC Resources 14-29
- Agenda 14-30
- Creating a Multi Data Source 14-31
- Configuring a Multi Data Source 14-33
- Multi Data Source WLST Example 14-34
- Managing Multi Data Source Members 14-35
- Quiz 14-36
- Summary 14-38
- Practice 14-2 Overview: (Optional) Creating and Using a Multi Data Source 14-39
- Practice 14-3 Overview: (Optional) Creating and Using a Proxy Data Source 14-40

15 Diagnostic Framework

- Objectives 15-2
- Agenda 15-3
- WebLogic Diagnostics Framework (WLDF) 15-4
- WLDF Architecture 15-5
- Diagnostic Archives 15-6
- Configuring Server Diagnostic Archives 15-7
- Diagnostic Modules 15-8
- Dynamic Diagnostic Modules 15-10
- Resource Descriptors 15-11
- Creating a Diagnostic Module 15-12
- WLST: Example 15-13
- WLST Commands for WLDF 15-14
- Quiz 15-15
- Agenda 15-16
- What Is a Diagnostic Image? 15-17
- Capturing a Server Diagnostic Image 15-18
- WLST: Example 15-19
- Quiz 15-20
- Agenda 15-21
- What Is a Harvester? 15-22
- Metric Collectors 15-23
- Configuring a Metric Collector 15-24
- WLST: Example 15-25
- Quiz 15-26
- Agenda 15-27
- Policies and Actions 15-28
- Policy Types 15-30
- Configuring Policies 15-31
- Configuring Actions 15-33
- Elastic Actions and Scaling 15-36
- Scale up Action 15-37
- Scale down action 15-38
- Quiz 15-39
- Summary 15-40
- Practice 15-1 Overview: Using a Built-in Diagnostic Module 15-41

16 WebLogic and Coherence Integration

- Objectives 16-2
- Agenda 16-3
- Oracle Coherence: Overview 16-4

Agenda	16-6
Types of Session Persistence	16-7
Coherence*Web	16-8
Coherence*Web and WebLogic Clusters	16-9
Coherence*Web Session Failover	16-10
Configuring Coherence*Web in WebLogic	16-11
Quiz	16-12
Practice 16-1 Overview: Configuring Coherence*Web	16-13
Agenda	16-14
Coherence and WebLogic Server	16-15
WebLogic Managed Coherence Servers Operations	16-16
What Is a Grid Archive (GAR)?	16-17
Coherence Application Deployment on WebLogic	16-18
Coherence Container: Benefits	16-19
Coherence Cluster	16-20
Managed Coherence Server	16-21
Quiz	16-23
Summary	16-24
Practice 16-2 Overview: Configuring Managed Coherence Servers	16-25

17 Application Staging and Deployment Plans

Objectives	17-2
Agenda	17-3
Review: Java EE Applications	17-4
Review: Deployment	17-5
Agenda	17-6
Server Staging Modes	17-7
Staged Mode Example	17-8
No Stage Mode Example	17-9
External Stage Mode Example	17-11
Configuring the Staging Mode	17-13
Quiz	17-14
Agenda	17-15
Development to Test to Production	17-16
Examples of Deployment Descriptor Values That Can Change	17-17
Agenda	17-18
Java EE Deployment Descriptors	17-19
Annotations	17-20
appmerge and appc	17-21
Quiz	17-22
Agenda	17-23

What Is a Deployment Plan?	17-24
Using Deployment Plans for Different Environments	17-26
Deployment Plan Example	17-28
Staging Deployment Plans	17-31
Quiz	17-32
Agenda	17-33
Creating a Deployment Plan	17-34
Creating a New Deployment Plan	17-36
Using the Administration Console to Generate a Deployment Plan	17-37
Modifying and Saving Data to Create a New Plan	17-38
New Deployment Plan Shows Changed Values	17-39
Using an Existing Deployment Plan to Configure an Application	17-40
Using an Existing Deployment Plan	17-42
WLST createPlan	17-43
weblogic.PlanGenerator	17-44
Using Plan Generator	17-45
Oracle Enterprise Pack for Eclipse	17-46
Managing Deployment Plans	17-50
Summary	17-51
Practice 17-1 Overview: Creating and Using a Deployment Plan	17-52

18 Production Redeployment

Objectives	18-2
Agenda	18-3
HTTP Sessions and Redeployment	18-4
Agenda	18-5
Redeployment Strategies	18-6
Agenda	18-7
Application Availability	18-8
What Is Production Redeployment?	18-9
Advantages of Production Redeployment	18-11
Production Redeployment Process	18-12
Application Retirement	18-14
Review: Administration Channel	18-15
Administration Mode	18-16
Distributing a Versioned Application	18-17
Deploying in Administration Mode	18-18
Rolling Back to the Previous Version	18-19
Quiz	18-20
Agenda	18-22
Redeployment Process: Overview	18-23

Configuring Application Deployment Versioning	18-24
Deploying a New Version of an Application	18-26
Distributing and Starting a Versioned Application in Administration Mode	18-30
Transitioning a Versioned Application from Administration Mode to Active	18-33
Rolling Back a Versioned Application to a Previous Version	18-35
Quiz	18-36
Agenda	18-37
Requirements and Restrictions	18-38
Summary	18-40
Practice 18-1 Overview: Using Production Redeployment	18-41

19 Oracle Cloud

Agenda	19-2
What is Cloud?	19-3
What is Cloud Computing?	19-4
History – Cloud Evolution	19-5
Components of Cloud Computing	19-6
Characteristics of Cloud	19-7
Cloud Deployment Models	19-8
Cloud Service Models	19-9
Industry Shifting from On-Premises to the Cloud	19-13
Oracle IaaS Overview	19-15
Oracle PaaS Overview	19-16
Oracle SaaS Overview	19-17
Summary	19-18

20 Oracle Java Cloud Service Overview

Objectives	20-2
Introducing Java Cloud	20-3
Java Cloud Service: Three Options	20-4
Java Cloud Service Main Use Cases	20-5
Java Cloud Service Feature: Provisioning	20-6
Java Cloud Service Feature: Patching	20-7
Java Cloud Service Feature: Backup / Restore	20-8
Java Cloud Service Feature: Scaling	20-9
Oracle Coherence Option: Data Caching & Scaling	20-10
Oracle Coherence Option: Your Cloud Data	20-11
How You Interact with Java Cloud Service	20-12
Speaking of Dev Environments... Developer Cloud Service	20-13
Java Cloud Service On-Premises!	20-14
Summary	20-15

A WebLogic Optimizations for Exalogic

- What Is Exalogic? A-2
- What Is InfiniBand? A-4
- Exalogic Networks A-5
- Exalogic Machine Topology A-6
- InfiniBand Networking Concepts A-7
- Default Compute Node Network A-8
- Recommended FMW Topology on Exalogic A-9
- Exalogic Shared Storage Concepts A-10
- Recommended FMW Storage on Exalogic A-11
- WebLogic Server Optimizations: Overview A-12
- Creating a Cluster Replication Channel A-13
- Using SDP for Replication A-14
- Using Multiple Ports for Replication A-15
- Other Replication Optimizations A-16
- Oracle Exabus A-17
- Other WebLogic Optimizations A-18
- Enabling Other Optimizations A-19
- What Is Exadata? A-20
- Exadata InfiniBand Connectivity A-21
- Enabling SDP for Exadata Connectivity A-22

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Sean Tay (taysiangmengsean@gmail.com) has a non-transferable
license to use this Student Guide.

1

Course Introduction

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Course Objectives

After completing this course, you should be able to:

- Perform a rolling upgrade of a WebLogic installation
- Configure domain templates and domains
- Configure automatic recovery from a machine crash
- Write scripts using the WebLogic Scripting Tool (WLST)
- Write script which use REST and WebLogic REST support
- Configure SSL for WebLogic servers
- Create and use deployment plans to move applications from one environment to another
- Configure and deploy Java EE shared libraries (optional)
- Configure and deploy two versions of an application simultaneously



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

In addition to these high-level terminal objectives, each lesson has a lower-level set of enabling objectives.

Course Objectives

- Configure work managers to control server requests
- Manage JDBC data sources
- Configure a JDBC multi and proxy data sources
- Create users, groups, roles, and policies in WebLogic's embedded LDAP
- Configure and manage simple Domain Partitions
- Back up and recover data from various failures
- Configure automatic JTA service migration
- Configure WebLogic Diagnostic modules
- Configure Coherence features that are integrated with WebLogic



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Target Audience

- This course is intended for WebLogic Server administrators.
- Learners should have completed *Oracle WebLogic Server: Administration I* or have equivalent experience.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

If you are concerned whether your background and experience match that of the target audience, ask the instructor.

Course Prerequisites

Required

- Basic knowledge of UNIX user-level commands and desktop navigation
- Basic familiarity with Extensible Markup Language (XML) concepts
- Basic TCP/IP networking knowledge of client/server concepts

Suggested

- Basic knowledge of Java Enterprise Edition (Java EE) concepts and constructs including Servlet, JavaServer Pages (JSP), Enterprise JavaBeans (EJB), and Java Message Service (JMS)



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

These prerequisites can be met by Oracle courses, or by other means of experience and training.

Introductions and Setting Expectations

Introduce yourself:

- Name
- Your company and role
- Your experience with WebLogic Server 12c and/or prior releases
- Anything specific you are looking for in this class



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Course Schedule

Day One

1. Introduction
2. WebLogic Server Review
3. Upgrading WebLogic Server
4. Creating and Using Domain Templates

Day Two

5. WebLogic Server Startup and Crash Recovery
6. WebLogic Scripting Tool (WLST)
7. WebLogic Server and REST



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This schedule is approximate and subject to change.

Course Schedule

Day Three

- 8. Secure Sockets Layer (SSL)
- 9. Shared Libraries
- 10. Work Managers
- 11. Security Realm

Day Four

- 12. High Availability, Migration, and Disaster Recovery
- 13. Domain Partitions
- 14. Managing Data Sources



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Course Schedule

Day Five

- 15. Diagnostic Framework
- 16. WebLogic and Coherence Integration
- 17. (Optional) Deployment Plans
- 18. (Optional) Production Redeployment



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Course Practices

- Each topic is reinforced with a hands-on practice.
- Most practices include a scripted solution to aid any students who encounter problems or run out of time.

Tasks

1. Obtain compute node IP addresses.

Solution Tasks

1. Access your <NODE1> compute node as **root** and execute the following:

```
cd /practices/Practice06_01/solution  
./createShares.sh <STORAGE1> <STUDENTID>
```



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Classroom Guidelines

- The instructor starts each session at the scheduled time.
- Do ask questions, but be respectful of the current topic and the interests of other students.
- Ensure that cell phones are silent.



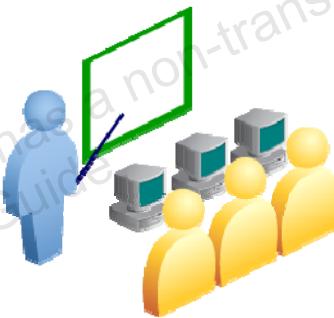
ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

These guidelines enable you to get maximum benefit from the course.

Facilities in Your Location

- Enrollment/Registration/Sign in
- Badges
- Parking
- Phones
- Internet
- Restrooms
- Labs
- Lunch
- Kitchen/Snacks
- Hours
- Materials (paper, pens, and markers)



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Contact your instructor or the education coordinator for site-specific information. This may not be applicable for a Live Virtual Class (LVC).

For More Information

Topic	Website
Education and Training	http://education.oracle.com
Product Documentation	http://www.oracle.com/technology/documentation
Product Downloads	http://www.oracle.com/technology/software
Product Articles	http://www.oracle.com/technology/pub/articles
Product Support	http://www.oracle.com/support
Product Forums	http://forums.oracle.com
Oracle Learning Library, Product Tutorials and Demos	http://www.oracle.com/oll



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

These materials are not intended to be a complete reference for all WebLogic Server JMS topics and features. After you complete the course, Oracle offers a variety of resources that you can use to obtain additional information.

Related Training

Course Title

Oracle WebLogic Server: Administration I

Oracle WebLogic Server: JMS Administration

Oracle WebLogic Server: Performance Tuning Workshop

Oracle WebLogic Server: Troubleshooting Workshop



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Note that some of the courses listed in the slide may be available as traditional or virtual classroom training, whereas others may be self-paced, online offerings.

WebLogic Server Review

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Define the terms:
 - Domain
 - Administration server
 - Managed server
 - Cluster
 - Node Manager
- List important WebLogic Server administrative topics



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Agenda

- WebLogic Server Domain and Resources
- Review WebLogic Administrative Topics



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Oracle WebLogic Server: Overview

Oracle WebLogic Server:

- Is a Java EE 6.0 application server
- Hosts Java EE applications
- Provides clustering for load balancing and high availability
- Offers an extensible security realm for authentication, authorization, and so on
- Is the platform on which the “Java components” of Oracle Fusion Middleware run



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Oracle WebLogic Server is a scalable, enterprise-ready, Java Enterprise Edition application server. WebLogic Server enables enterprises to deploy mission-critical applications in a robust, secure, highly available, and scalable environment. These features enable enterprises to configure clusters of WebLogic Server instances to balance workload, provide extra capacity, and fail over in case of hardware or other failures.

Extensive security features protect access to services and keep enterprise data secure.

Oracle Fusion Middleware is a collection of standards-based products that spans a range of tools and services: from Java EE, to integration services, business intelligence, and collaboration. Fusion Middleware products are organized into two general categories: Java components and system components. Java components generally are deployed to WebLogic Server as one or more Java Enterprise Edition applications and a set of resources. System components are not deployed as Java applications. Instead, a system component is managed by the Oracle Process Manager and Notification server (OPMN). Oracle HTTP Server is an example of a system component.

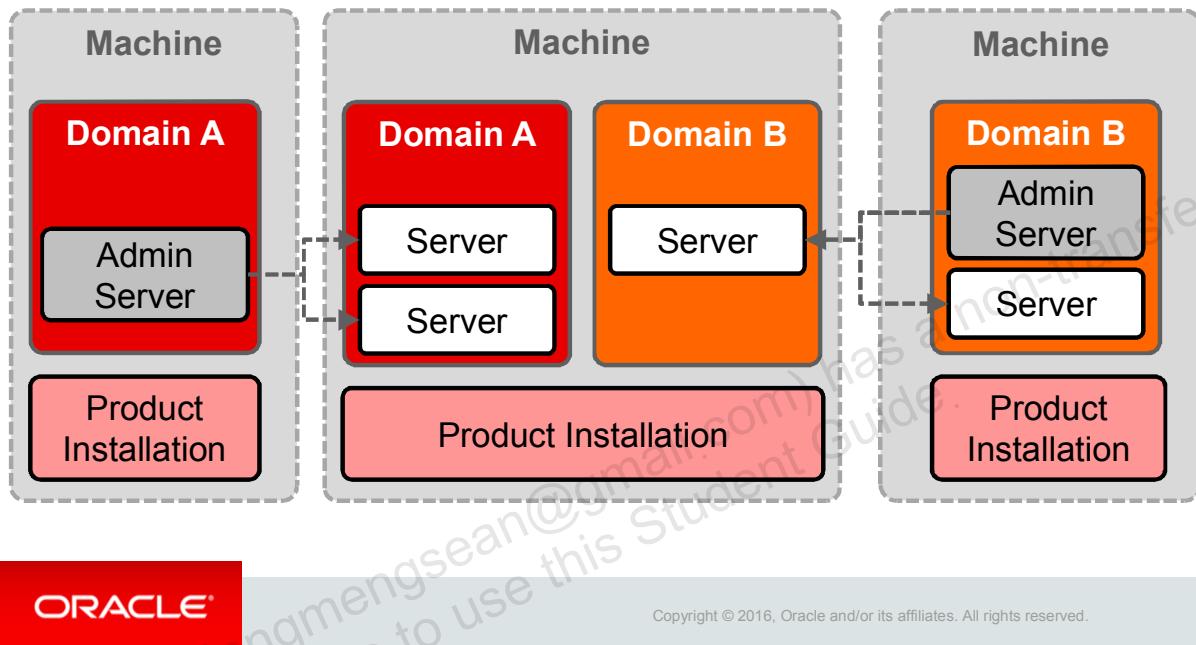
Some of the Fusion Middleware Java components include the following:

- **Oracle SOA Suite:** A set of infrastructure components for designing, deploying, and managing service-oriented architecture (SOA) applications. SOA Suite components run on top of WebLogic Server.
- **Oracle Service Bus:** An Enterprise Service Bus (ESB) that provides the infrastructure for service discovery, service provisioning and deployment, and governance. This service-infrastructure software adheres to the SOA principles of building coarse-grained, loosely coupled, and standards-based services.
- **Oracle WebCenter Suite:** An integrated set of components used for creating social applications, web portals, collaborative communities, and composite applications
- **Oracle Identity Management:** An enterprise identity management system that manages user access privileges within the resources of an enterprise. Some of its components run on WebLogic Server, for example, Oracle Directory Integration Platform. This is a Java EE application that enables you to synchronize data between different repositories and Oracle Internet Directory.

WebLogic Server Domain

WebLogic Server resources are grouped into logical domains.

- You can decide the number of domains and how they are organized.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

A *domain* is a collection of WebLogic Server resources. There are many different kinds of resources in a domain, including WebLogic Servers, deployed applications, clusters, security providers, and Java Message Service and Java Database Connectivity elements.

How many domains you have and how you organize them is completely left to you. For example, domains can be organized by a logical division of types of applications, by physical location of hardware, by size and number of administrators, and so on.

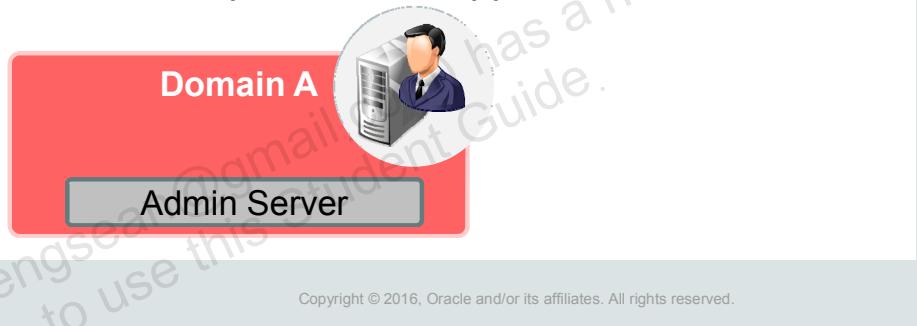
All domains contain a special server called the *administration server*. You use the administration server to configure and manage all the domain resources. Any other WebLogic Servers in the domain are called *managed servers*.

In most domains, the applications are deployed to the managed servers. The administration server is used only for domain configuration and management.

A single WebLogic Server product installation can be used to create and run multiple domains, or multiple product installations can be used to run a single domain. You can decide how domains are defined. You can define multiple domains based on different system administrators' responsibilities, application boundaries, or geographical locations of the machines on which servers run.

Administration Server

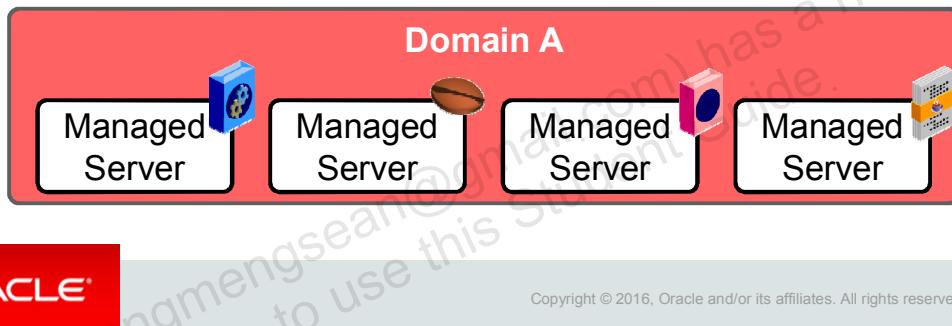
- A domain must have exactly one instance of WebLogic Server acting as the *administration server*. An administration server is part of exactly one domain.
- The administration server is:
 - The central point through which you configure and manage all domain resources
 - Solely in charge of the domain's configuration. It distributes configuration changes to other servers in the domain.
 - An instance of WebLogic Server and, therefore, a fully-functional Java Enterprise Edition application server



Because an administration server is an instance of WebLogic Server, it can perform any task of a Java Enterprise Edition application server. Applications can be deployed and run on the administration server. For simplicity, often a development-time domain will contain only the administration server. Developers deploy and test their applications on the administration server.

Managed Servers

- A domain can have zero or more *managed servers*.
- A managed server:
 - Is managed by the administration server
 - Is an instance of WebLogic Server and, therefore, a fully-functional Java Enterprise Edition application server
 - Is where your Java Enterprise Edition applications run
 - Web applications, EJBs, web services, enterprise applications
 - Can be clustered with other cooperating managed servers for availability, scalability, and automatic failover



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

In almost all domains, the administration server is not the only server defined in the domain. Other servers are also defined. These servers are called managed servers, because they are managed by the administration server.

A company's web applications, EJBs, web services, and other resources are deployed and run on the managed servers. That leaves the administration server free for configuration and management purposes.

For scalability, availability, and failover (when one server fails, requests are automatically sent to another server) managed servers can be placed together in a cluster.

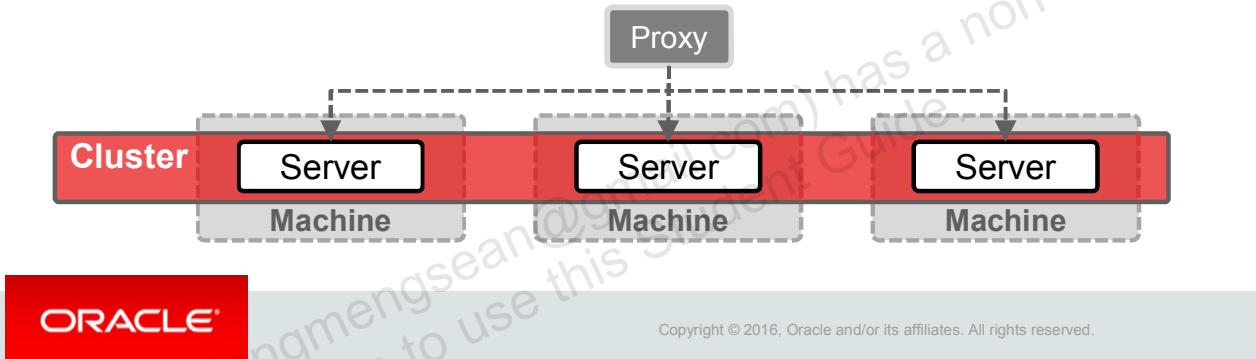
Machines and Clusters

A *machine*:

- Is defined within a domain to represent physical hardware
- Is required by the Node Manager and used by clusters
- Has managed servers assigned to it

A *cluster*:

- Has multiple managed servers running cooperatively in it, which provides for failover
- Requires a cluster proxy that provides load balancing



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

A machine definition is used to associate a computer with the managed servers that it hosts. The Node Manager is always defined within the context of a machine. Machine definitions, and which servers are assigned to them, are also used by a clustered managed server in selecting the best location for storing replicated session data.

A cluster is a collection of multiple managed servers within a single domain running simultaneously and cooperatively to provide increased scalability and reliability. In a cluster, resources and services are deployed identically to each server, allowing for failover and load balancing. The session state of one clustered server is replicated on another server in the cluster. When a server fails, another server in the cluster takes over and retrieves the replicated data. No information is lost and customers do not realize that a different server is now fulfilling their requests. Clustered servers communicate with one another in two main ways: sending updates to their “backup server” when session state changes, and through cluster “heartbeats.” Each clustered server sends out a signal periodically to indicate that it is still viable. If a member of the cluster misses three heartbeats, that server is considered “failed.”

A cluster is always fronted by a proxy for web applications, which could be a web server, a hardware load balancer, or an instance of WebLogic Server. The proxy provides load balancing and enables failover by avoiding failed servers. A proxy is not required for EJB or JMS clients because they perform load balancing and failover using different techniques.

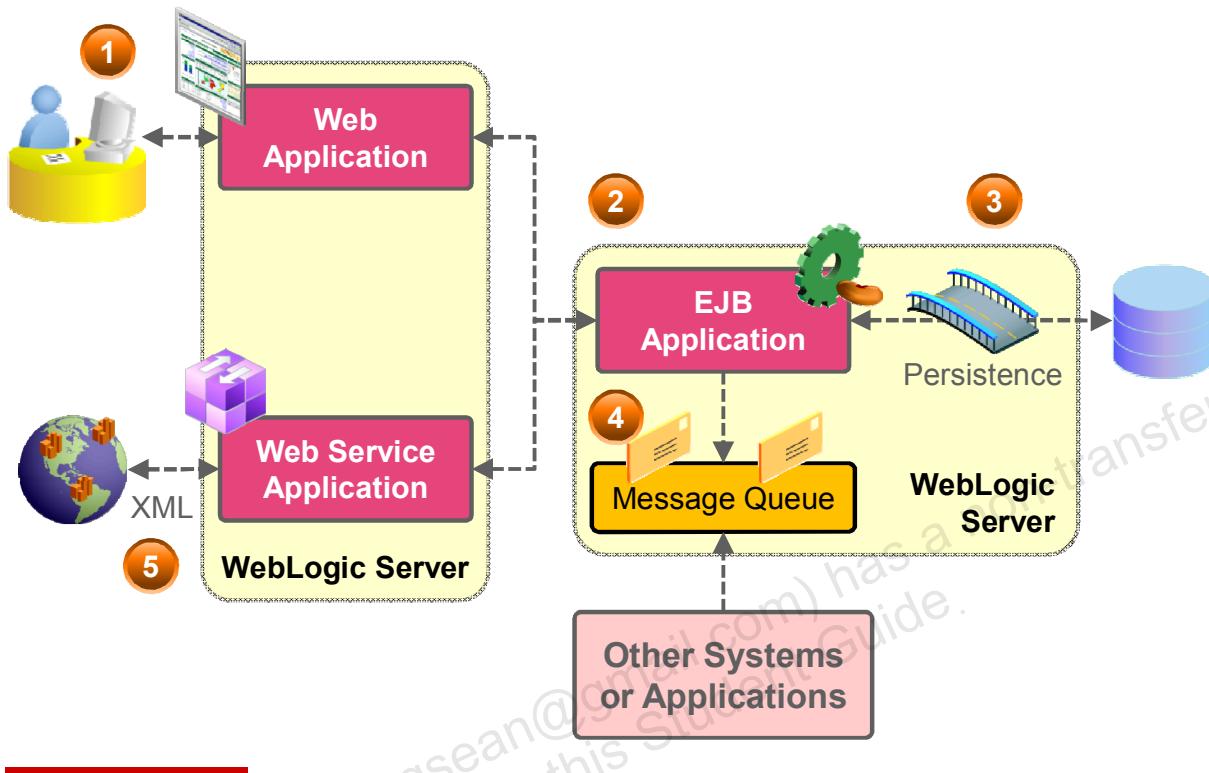
WebLogic Server Application Services

- Java Database Connectivity (JDBC) 
 - It is the API for accessing relational databases.
 - Data source objects are configured to provide database access.
- Java Message Service (JMS) 
 - It is the API and implementation of an enterprise messaging system.
 - Multiple resources must be configured in WebLogic Server for JMS.
- Java Transaction API (JTA) 
 - When transactions need to span resources, WebLogic Server can act as the transaction manager.

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic Server Application: Example



1. Users interact with a web application by using a browser. The web application is responsible for rendering the website and for capturing user input through buttons, forms, links, and so on. It is possible that the web application contains all the necessary business logic to perform the tasks that users request.
2. In this example, however, the web application accesses Enterprise JavaBeans (EJBs) to perform the business logic. These EJBs can be located on the same server as the web application or on a different server, as shown in this example.
3. Some of the EJBs shown include a persistence mechanism. They are writing newly placed orders to a relational database.
4. After the order is written to the database, an EJB uses the Java Message Service (JMS) to asynchronously communicate with other applications so that those applications can also process the order.
5. To expose the business logic of this application in a standard way to other applications, both within your organization and beyond, a web service is used. XML-based web services can be accessed by both Java and non-Java applications and are a cornerstone of service-oriented architecture.

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Agenda

- WebLogic Server Domain and Resources
- Review WebLogic Administrative Topics



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Start Scripts

WebLogic Server servers are started using shell scripts:

- Administration server:

```
cd $DOMAIN_HOME  
./startWebLogic.sh
```

- Managed servers:

```
cd $DOMAIN_HOME/bin  
./startManagedWebLogic.sh <server_name> <admin_url>
```

Both scripts source setDomainEnv to set the environment.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Navigate to the directory in which you created the domain. By default, this directory is `MW_HOME/user_projects/domains/DOMAIN_NAME`, where `DOMAIN_NAME` is the root directory of the domain. (The name of this directory is the name of the domain.)

Run one of the following scripts to start the administration server:

- `bin/startWebLogic.sh` (UNIX and Windows. On Windows, this script supports the MKS and Cygnus BASH UNIX shell emulators.)
- `bin\startWebLogic.cmd` (Windows)

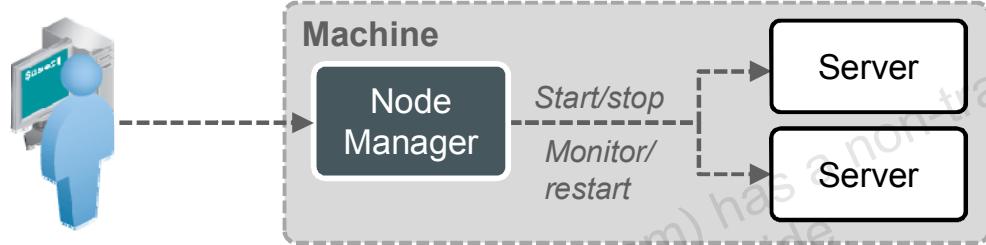
Run one of the following scripts to start a managed server:

- `bin/startManagedWebLogic.sh` (UNIX and Windows)
- `bin\startManagedWebLogic.cmd` (Windows)

Note: The `startWebLogic.sh` script located in the `DOMAIN_HOME` folder invokes the `startWebLogic.sh` script in the `DOMAIN_HOME/bin` folder. You can alternatively run the script in the `bin` folder directly. Each of the start scripts sources the `DOMAIN_HOME/bin/setDomainEnv`. [sh | cmd] script to set the environment.

Node Manager

- Is a separate process that accepts remote commands to start, stop, or suspend servers on its machine
- Monitors server availability and can restart failed servers
- Can be used to migrate servers on a failed machine to another machine



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Server instances in a WebLogic Server production environment are often distributed across multiple domains, machines, and geographic locations. The Node Manager is a WebLogic Server utility that enables you to start, shut down, and restart both administration server and managed server instances from a remote location. Although a Node Manager is optional, it is recommended if your WebLogic Server environment hosts applications with high-availability requirements. A Node Manager process runs on a particular machine.

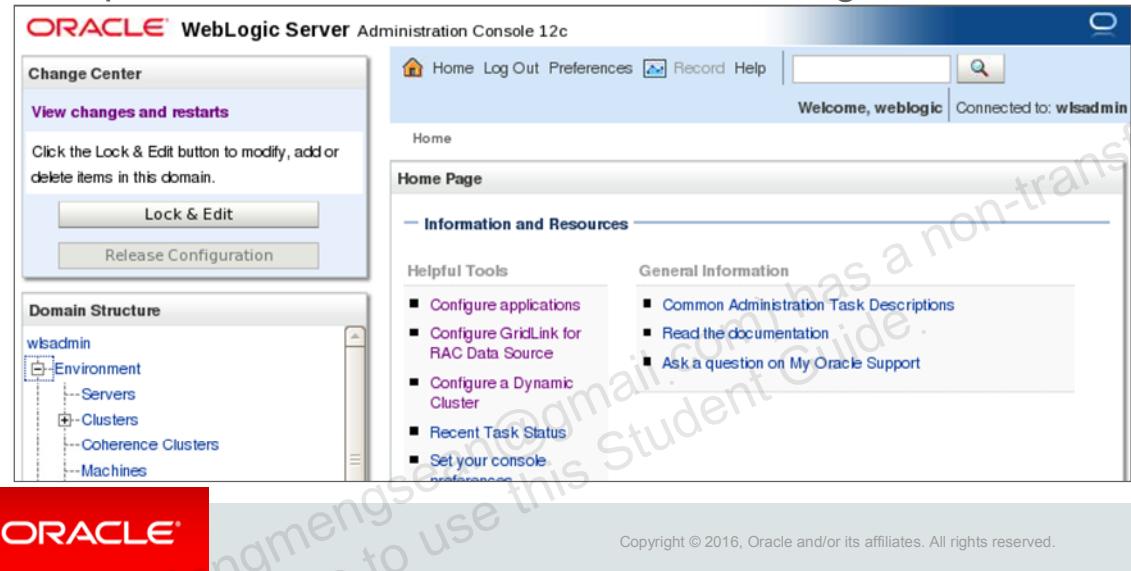
There are two versions of Node Manager: the Java-based one that runs on any platform WebLogic Server runs on and the script-based one that runs only on *nix operating systems. The Java-based Node Manager is recommended.

If a server that was started by using the Node Manager fails, the Node Manager can be set to automatically restart it. If the Node Manager fails or is explicitly shut down, upon restart, it determines the servers that were under its control when it exited. The Node Manager can restart any failed servers as needed.

The WebLogic Server Administration Console can be used to issue commands to Node Managers running on remote machines. The WebLogic Scripting Tool (in offline mode) also serves as a Node Manager command-line interface.

WebLogic Tools: Administration Console

- The Oracle WebLogic Server Administration Console is a web browser-based tool for configuring, administering, and monitoring the resources of a domain.
- The console application runs on the administration server.
- It is part of the normal installation of WebLogic Server.



The Oracle WebLogic Server Administration Console (admin console) is a web browser-based graphical user interface that can be used to manage a WebLogic Server domain. The Administration Console application runs on the administration server. The Administration Console can be used to:

- Configure, start, and stop instances of WebLogic Server
- Configure clusters
- Configure database connectivity (JDBC)
- Configure messaging (JMS)
- Configure WebLogic Server security
- Deploy applications
- Monitor server and application performance
- View server and domain log files

WebLogic Tools: WebLogic Scripting Tool (WLST)

- WLST is a scripting tool for configuring, administering, and monitoring the resources of a WebLogic Server domain.
- It is part of the normal installation of WebLogic Server.
- WLST can perform all the tasks (and more) available in the Administration Console.
- It can run commands one at a time, or from files (scripts).
- It is based on the Jython programming language (the Java implementation of Python).

```
# Modify these values as necessary
username = "weblogic"
password = "Welcome1"
url = "t3://myadminhost:7001"
# Connect to the admin server
connect(username, password, url)
```



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

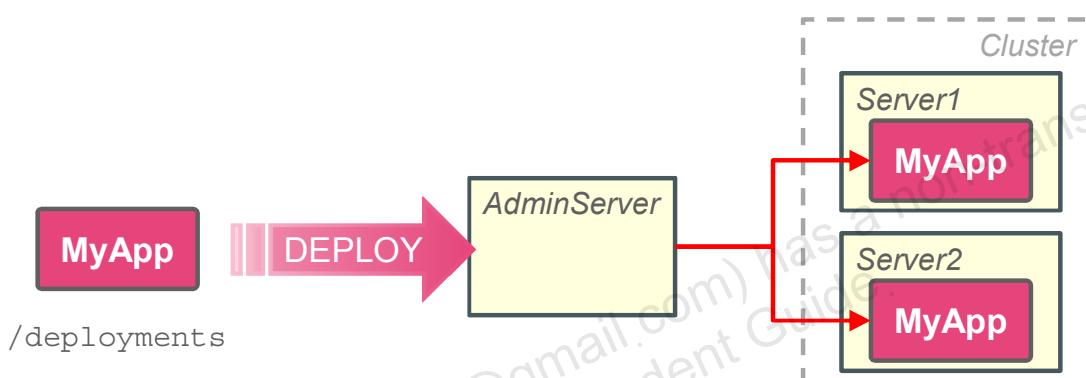
The WebLogic Scripting Tool (WLST) is a command-line scripting environment that can be used to create, manage, and monitor WebLogic Server domains. It is based on the Java scripting interpreter, Jython. In addition to supporting standard Jython features, such as local variables, conditional execution, and flow control statements, WLST provides a set of commands that are specific to WebLogic Server.

WLST can be run interactively (one command at a time) or in script mode (running a file of commands). It also can be run online (connected to an administration server, which allows it to manage an active domain) or offline (accessing the configuration files of an inactive domain).

Deployment

Is the process of:

- Packaging an application to prepare it for deployment
- Configuring the application to run on the target environment
- Deploying the application to start servicing requests



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

- **Preparing applications and modules for deployment:** You can deploy applications either as archived files or as exploded archive directories. Oracle WebLogic Server also introduces the concept of an application installation directory, which helps you to organize the deployment files and the deployment configuration files for easy deployment using Oracle WebLogic Server tools. Before deploying an application, an administrator typically prepares the deployable modules by creating an application installation directory and copying the application archive file into the appropriate subdirectory.
- **Configuring the application or module for deployment to the Oracle WebLogic Server environment:** Administrators typically receive a new application (or a new version of an application) from their development team and must deploy the application to a staging or production environment that differs from the environments used during development and testing. Oracle WebLogic Server helps you to easily configure an application for a new target domain without having to manually edit the deployment descriptor files provided by development team. The configuration changes for a specific deployment environment are stored in a new configuration artifact, a deployment plan, that is stored and maintained independently of the deployment files provided by the development team.
- **Deploying the application to Oracle WebLogic Server:** After preparing both the deployment and configuration files, applications are distributed to the target servers in an Oracle WebLogic Server domain and made active for processing client requests.

Summary

In this lesson, you should have learned how to:

- Define the terms:
 - Domain
 - Administration server
 - Managed server
 - Cluster
 - Node Manager
- List important WebLogic Server administrative topics



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

3

Upgrading WebLogic Server

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to perform a rolling patch upgrade on a live WebLogic Server clustered environment.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Note

A rolling upgrade is a process of upgrading the WebLogic Server installation on the machines across your domain without disrupting the application or clients.

Agenda

- Upgrade Definition
 - Patch
 - Minor Upgrade
 - Major Upgrade
- Rolling Upgrade
- Rolling Upgrade Process



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Definition of Upgrade

An upgrade is any product patch or release that fixes issues or adds new features to the product. Upgrades are loosely categorized into the following types:

- Patch
- Minor release
- Major release



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Patch

A patch is a set of files that contain bug fixes and other enhancements that improve the functionality of a running WebLogic installation:

- Patches are installed using the Oracle Patch (OPatch) mechanism:
 - Patches are applied to upgrade the WebLogic Server installation files on each machine within a domain.
 - OPatch supports rolling patches back (downgrading).
- Patches support rolling upgrades.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Minor Upgrade

A minor upgrade consists of upgrading from one release of WebLogic Server to the next minor release within the same major release:

- Example: Upgrading from 12.1.1 to 12.1.2
- Minor release rolling upgrades are not officially supported in 12c.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Minor release upgrades used to be supported by WebLogic. This was made possible because the server was implemented to allow servers in the same cluster to run with different minor release version numbers. The release versioning has changed over time and this is no longer officially supported, but it may still work (unofficially).

Major Upgrade

A major upgrade consists of upgrading from one major release of WebLogic Server to another major release:

- Example: 9.x to 10.x
- Major release rolling upgrades are not officially supported in 12c.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Major upgrades have never been supported by WebLogic. The changes made in a major release are often too different for servers in the same cluster to run using different versions. Examples of this include going from WebLogic version 8.1 to 9.0, which made major web service and JMX architectural changes.

Quiz



Which of the following upgrade types are officially supported for WebLogic rolling upgrades?

- a. Domain
- b. Minor
- c. Major
- d. Patch

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: d

Agenda

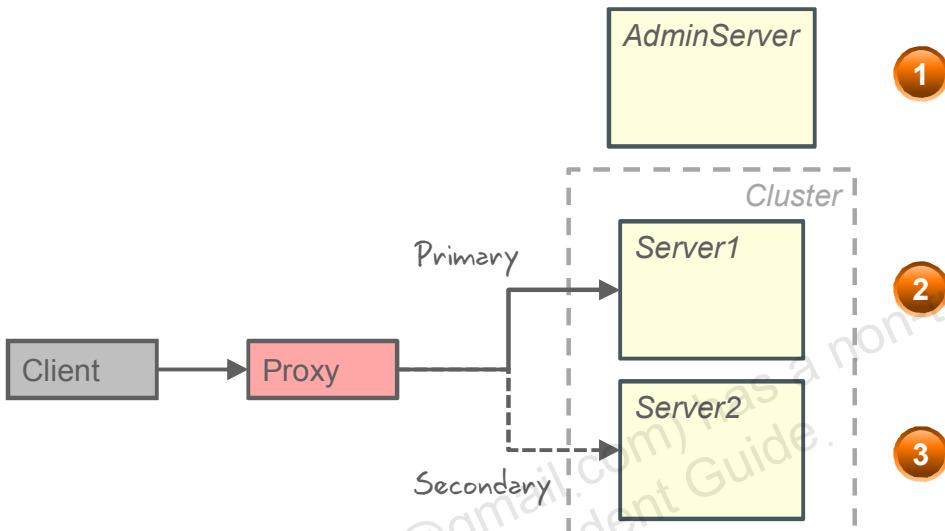
- Upgrade Definition
- Rolling Upgrade
 - What is a Rolling Upgrade?
 - Multiple Installation and Domain Locations
 - Leverage WebLogic Clusters to Avoid Down Time
- Rolling Upgrade Process



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

What Is a Rolling Upgrade?

A rolling upgrade is the process of upgrading a running WebLogic Server cluster without shutting down the entire cluster or domain.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic supports the ability to upgrade a domain by performing the upgrade on one server at a time. This process allows the rest of the domain to function as each individual server is upgraded with a patch of the product. This process accounts allowing active client sessions to continue functioning properly if session replication is used. This allows the upgrade process to occur unnoticed by connected users of the application. The end result is a zero down time upgrade process.

Note: Although minor releases are not officially supported at the time of writing this course, always refer the latest documentation to see what is currently supported.

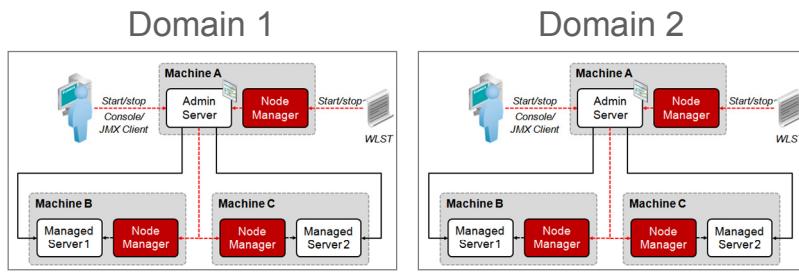
The rolling upgrade process for the example illustrated in the slide involves:

1. Upgrading the administration server first. A domain can function properly only when the administration server is running the latest version of WebLogic.
2. Upgrading the server1 managed server
3. Upgrading the server2 managed server

Multiple Installation and Domain Locations

If multiple domains are deployed on the same machines, they can use:

- The same WebLogic installation
- Separate WebLogic installations



Both domains can use the same WebLogic installations or separate installations on all machines.

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Most production systems do not have multiple domains sharing the resources of the same machines. However, in cases where they are deployed on the same machines, they can optionally use the same installation of WebLogic Server or separate installations of WebLogic Server. There are advantages and disadvantages to both approaches.

Using the same installation of WebLogic Server

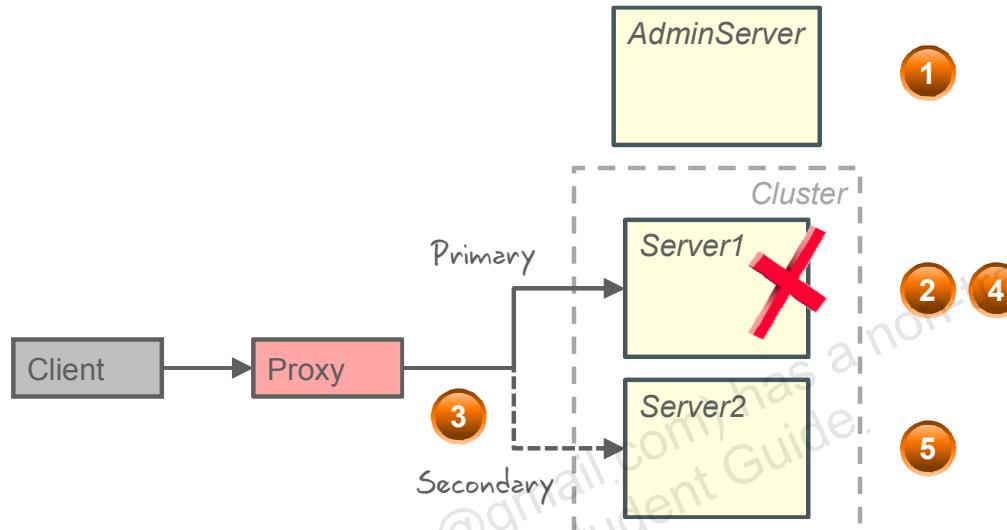
- Ensures that when the product is upgraded, all domains that use that installation are running the same version of software
- Makes the rolling upgrade process more complicated because every server for every domain on a machine getting upgraded must be shut down before upgrading the installation

Using separate installations of WebLogic Server

- Allows independent administration of each domain, including rolling upgrades
- Simplifies administration of rolling upgrades because only one domain needs to be considered at a time
- Separates the patch requirements of each domain so that one domain is not forced to upgrade because another domain requires a patch

Leverage WebLogic Clusters to Avoid Down Time

Using a cluster for your applications with WebLogic ensures that while upgrading, the clients using deployed applications continue to function.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

A true rolling upgrade can occur only in a clustered WebLogic environment. This is because the purpose of a rolling upgrade is to provide zero down time for the application during the upgrade process. The only exception to this rule is upgrading the AdminServer, which causes an outage only in performing certain administrative tasks until the AdminServer is restarted. An application that is deployed on a cluster is still available for clients when a single member of the cluster is shut down.

Applications that use WebLogic maintain HTTP and possibly stateful EJB sessions. When a server is shut down, all sessions are lost. If the session is accessed again, a new secondary session is created within the domain. During a rolling upgrade, only one server is upgraded at a time, so any clients that lose their sessions are automatically backed up by their secondary sessions. So clients that maintain activity during a rolling upgrade process continue to maintain a valid session throughout the upgrade process. Clients that are idle during the process when the servers containing their primary and secondary sessions are shut down, lose their session state and must reestablish their sessions with the application.

The following is an example of how this works with HTTP sessions:

1. The administration server is upgraded. This affects only administrative clients.
2. The `server1` managed server is shut down. All primary and secondary sessions stored on this server are lost.
3. Clients continue to use the application and their sessions fail over to their secondary backup copies.
4. You upgrade `server1` and start it.
5. Clients continue to use the secondary (which has been promoted to primary) copy of their sessions. Now that `server1` is available again, secondary sessions are replicated to `server1`. The process then continues by shutting down, upgrading, and restarting `server2`. When this occurs, the active client session fails over to `server1` to avert any client down time.

Quiz



The primary purpose of performing a rolling upgrade on a domain is to upgrade WebLogic while a domain:

- a. Is shutdown to avoid downtime
- b. Is running to avoid downtime
- c. Is restarting to avoid downtime
- d. None of the above

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: b

Agenda

- Upgrade Definition
- Rolling Upgrade
- Rolling Upgrade Process
 - Backup
 - Shutdown
 - Upgrade
 - Restart



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Rolling Upgrade Process: Overview

The rolling upgrade process involves the following high-level tasks:

1. Backing up whatever is getting changed:
 - WebLogic Server installation
 - Domains
 - More
2. Shutting down the servers on one machine
3. Upgrading the WebLogic installation or domain on that machine
4. Restarting the servers on that machine
5. Repeating the process on each machine in the domain



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Tip: You should be mindful of the session timeout for your applications. If you wait between upgrading each machine in your domain for the session timeout to expire, then your chances of allowing users to use the application and have sessions fail over to other servers increases. Only users that have not used the application for over the session timeout will experience disruption, which they would expect anyway because they haven't used the application for a long period of time.

Backup

Back up anything that you are going to change as part of the upgrade process:

1. WebLogic installation files
2. Domain files

This ensures that you can return to a known working state quickly if anything goes wrong.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

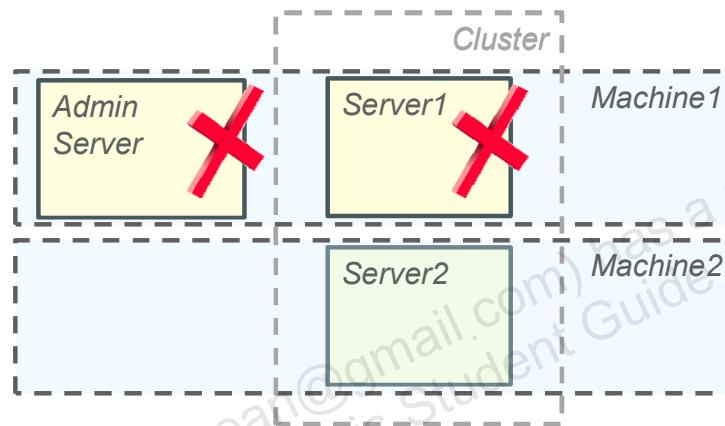
Note

Even if you are not going to change the domain, Oracle recommends backing up your domain because files can potentially change when running under an effectively newer version of WebLogic. However, it is not required to back up the WebLogic installation if only domain changes are made.

Shutdown

Orchestrate an orderly shutdown of servers in the domain:

1. Shut down the administration server first.
2. If any managed servers are on the same machine as the administration server, then shut them down.
3. Shut down servers only on one machine at a time!



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

All the servers sharing the same WebLogic installation files or domain configuration files must be shut down at the same time before performing any upgrades. Servers on other machines must not be shut down because they are required to be available to support failover for client applications. This is a vital part of the rolling upgrade process that minimizes or eliminates client down time.

Note: Shut down is a forced shut down; otherwise, the shutdown process will wait until all client sessions have ended prior to shutting down. The rolling upgrade process involves shutting servers down while clients are still running to allow the production redeployment mechanism to manage the graceful transition to the new version of the application.

Upgrade

Apply changes to the domain and installation:

1. Apply a patch using OPatch.
2. Roll back a patch using OPatch.
3. Update classes in your installation or domain.
4. Realize domain configuration changes that require a restart.



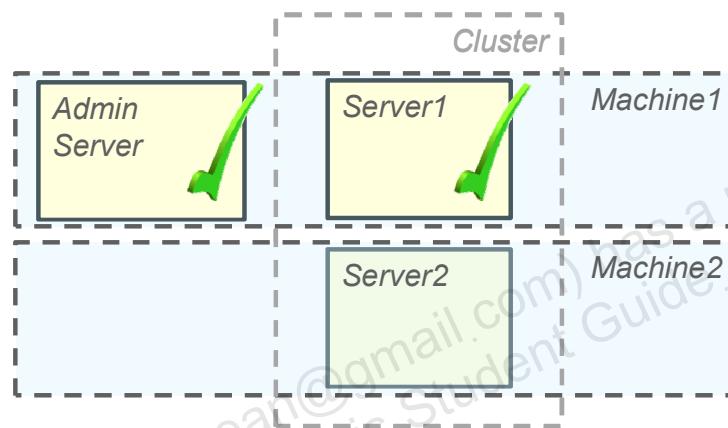
Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Although the main focus of this lesson is performing a rolling upgrade using OPatch to apply official Oracle patches to a WebLogic installation, there are other changes you can make to your domains that may require restarting the servers in your cluster. When this occurs, those changes may also be realized using a rolling upgrade approach to avoid system down time.

Restart

Orchestrate an orderly startup of servers in the domain:

1. Start up the administration server first.
2. If any managed servers are on the same machine as the administration server, start them up.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

After everything has been upgraded on the current machine, you restart all the servers on that machine. You start with the administration server, followed by any managed servers. The administration server is upgraded first because a domain requires that the administration server is always running the latest version of software in the domain. This ensures that any configuration or feature changes are supported. WebLogic also allows servers in a cluster to temporarily run with slightly different versions, which enables the rolling upgrade feature.

After all the servers of the current machine are running, you continue the rolling upgrade process by performing these steps on the next machine until the entire domain is upgraded.

Quiz



Which of the following are two reasons why you shut down servers on only one machine at a time during a rolling upgrade?

- a. Allowing servers to stabilize during an upgrade
- b. Avoiding disruption of client sessions
- c. Avoiding too many session objects for clients
- d. Allowing clients to failover to other servers

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: b, d

Summary

In this lesson, you should have learned how to perform a rolling patch upgrade on a live WebLogic Server clustered environment.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Practice 3-1 Overview: Setting Up the Course Practice Environment

This practice covers the following topics:

- Setup the intial course environment
- Verify that a WebLogic domain is functioning properly



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Practice 3-2 Overview: Performing a Rolling Upgrade

This practice covers the following topics:

- Running a Shopping Cart application that is deployed on a cluster that hosts two managed servers
- Running Oracle HTTP Server to proxy requests to the cluster
- Shutting down, upgrading, and starting up one server at a time while running the Shopping Cart application
- Verifying that the application continues to work as expected throughout the upgrade of the entire domain



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

4

Creating and Using Domain Templates

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Create a domain template by using the Template Builder
- Create an extension template by using the Template Builder
- Create a domain using a template
- Extend a domain using a template



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Agenda

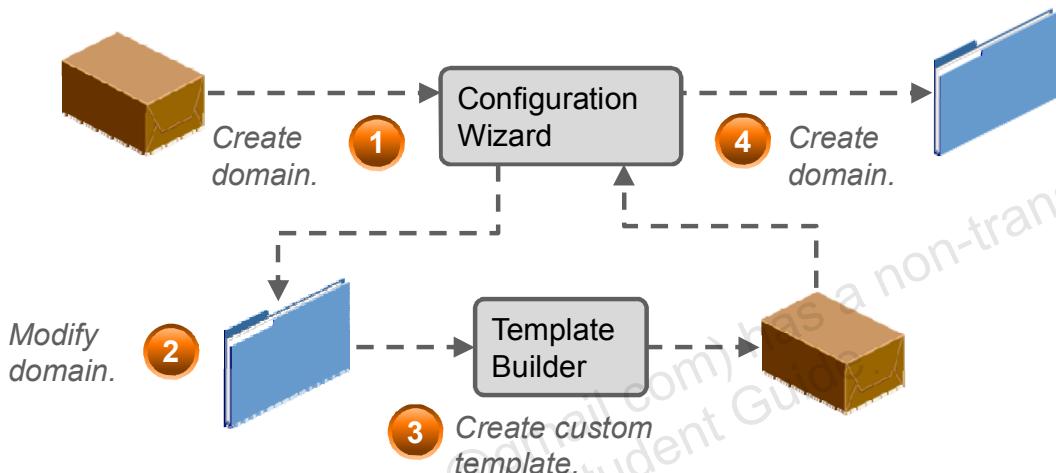
- Review
- Domain Template Concepts
- Extension Template concepts
- Review: FMW Templates
- Custom Templates



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Domain Template Review

- The Fusion Middleware Configuration Wizard creates WebLogic domains based on templates.
- Using custom templates, you can distribute a baseline domain configuration across multiple projects.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

A domain is the basic administration unit of WebLogic Server. It consists of one or more WebLogic Server instances, and logically related resources and services that are managed collectively, as one unit. In addition to infrastructure components, such as servers and clusters, a domain defines application deployments, supported application services (such as database and messaging services), security options, and physical host machines.

The Configuration Wizard guides you through the process of creating a domain for your target environment by selecting the product components that you want to include in your domain, or by using domain templates. If required, you can also customize the domain to suit your environment by adding and configuring managed servers, clusters, and machine definitions, or customizing predefined JDBC data sources and JMS file store directories.

A domain template defines the full set of resources within a domain, including infrastructure components, applications, services, security options, and general environment and operating system options. You can create this type of template from an existing domain by using the Domain Template Builder, WebLogic Scripting Tool (WLST), or the pack command-line tool. Subsequently, you can create a domain based on the template by using the Configuration Wizard or WLST.

The product installation includes a set of predefined domain and extension templates. This set includes the base WebLogic Server domain template and various extension templates that allow you to add component features and samples to the base domain.

Agenda

- Review
- Domain Template Concepts
 - Template Contents
 - Template Exclusions
 - Script Replacement Variables
 - Template SQL Scripts
- Extension Template Concepts
- Review: FMW Templates
- Custom Templates

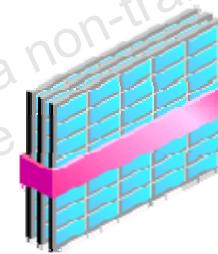


Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Template Contents

A domain template contains some combination of the following:

- Domain configuration (`config.xml`) and supporting resource modules (JDBC, JMS, diagnostics)
- Java Enterprise Edition (Java EE) applications and shared libraries
- Windows and UNIX server start scripts
- Windows Start menu entries
- Other custom folders and files



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Template Exclusions

Generated domain templates automatically exclude the following domain files:

- The contents of the domain's servers folder (logs, cached application resources, cached LDAP data, and so on)
- Locks and other temporary files



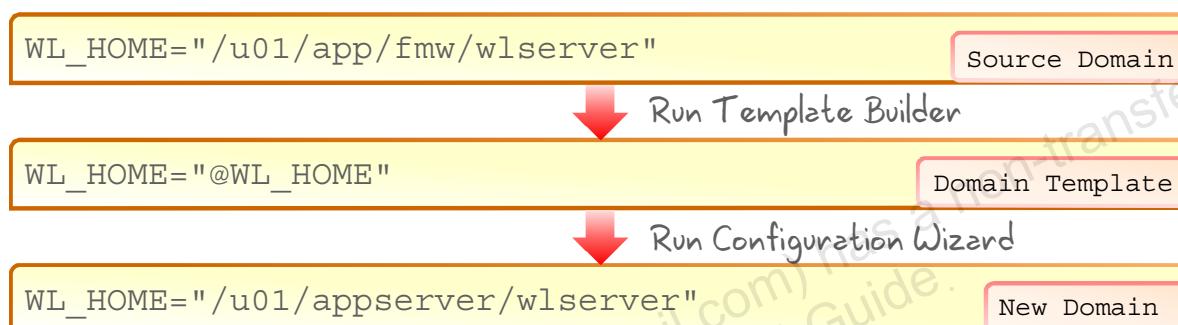
ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Some files are not included with a domain template. The files that are excluded include large files that can greatly increase the size of the template and files that are most likely not needed in any domains you will create from the template.

Script Replacement Variables

- All literal paths in standard server start scripts are automatically replaced with variables by the Domain Template Builder.
- The Configuration Wizard replaces these variables with paths that correspond to the local product and domain installation.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

When you are creating a template, you want the scripts and files that you are packaging with your template free of local domain environment settings and ready for use by the Configuration Wizard. The Domain Template Builder automatically updates any standard scripts included in a template, such as start scripts, by replacing hard-coded values for various domain environment settings with replacement variables. The Configuration Wizard can later replace these variables with new hard-coded values during the configuration of a new domain.

Custom scripts and other files are not automatically updated with replacement variables. The Domain Template Builder interface includes a script editor to help with this task. Select the file to edit, highlight the literal string to replace, and select from a list of defined variables to replace the string with. Commonly used variables include DOMAIN_NAME, DOMAIN_HOME, JAVA_HOME, and WL_HOME. The list of files for the Configuration Wizard to update along with the variables to replace is stored within the template as a file named stringsubs.xml.

Agenda

- Review
- Domain Template Concepts
- Extension Template Concepts
- Review: FMW Templates
- Custom Templates

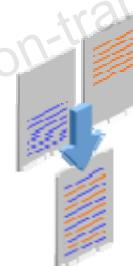


Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Extension Template: Concepts

An extension template:

- Can be created from an existing domain or from another domain template
- Includes a `config.xml` file whose contents are merged with the target domain's configuration
- Does not include definitions for servers, clusters, or machines (removed by the Template Builder)



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Extension templates define applications and services that can provide additional features for a domain, such as JDBC or JMS components. This type of template can be used to update an existing domain. The product installation includes a set of predefined domain and extension templates. This set includes the base WebLogic Server domain template and various extension templates that allow you to add component features and samples to the base domain. But you may also use the Template Builder tool to create your own custom extension templates as well.

Agenda

- Review
- Domain Template Concepts
- Extension Template Concepts
- Review: FMW Templates
- Custom Templates



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Fusion Middleware (FMW) Templates

- A simple, base WebLogic Server domain template is bundled with the product at <WL_HOME>/common/templates/wls.
- Other FMW products install additional extension templates:
 - To initialize domains with product-specific resources and applications
 - At <PRODUCT_HOME>/common/templates/applications



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

When additional Oracle FMW products have been installed, the Select Domain Source window in the Configuration Wizard will include check boxes that correspond to the components of these products. Each component is typically associated with a domain or an extension template.

When creating your domain, if you choose not to configure all the components in a given product suite, such as Oracle SOA Suite, you can add these components at a later date by extending your domain.

One of these product components is Java Required Files (JRF), which consists of those resources not included in the default Oracle WebLogic Server installation and which provides common functionality for other FMW products and application frameworks.

Quiz



Which two types of domain resources are NOT included in an extension template?

- a. Data Source
- b. Cluster
- c. Application
- d. JMS Module
- e. Server

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: b, e

Quiz



Which two of the following are NOT a feature of domain templates?

- a. SQL scripts
- b. Start menu entries
- c. Email notifications
- d. Variable replacement

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: a, c

Agenda

- Review
- Domain Template Concepts
- Extension Template Concepts
- Review: FMW Templates
- Custom Templates
 - Why use Custom Templates
 - Template Builder
 - Creating a Domain Template
 - Creating an Extension Template
 - Using a Custom Template with the Configuration Wizard
 - Post Domain Creation Tasks
 - Using a Custom Extension Template with the Configuration Wizard



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Why Use Custom Templates

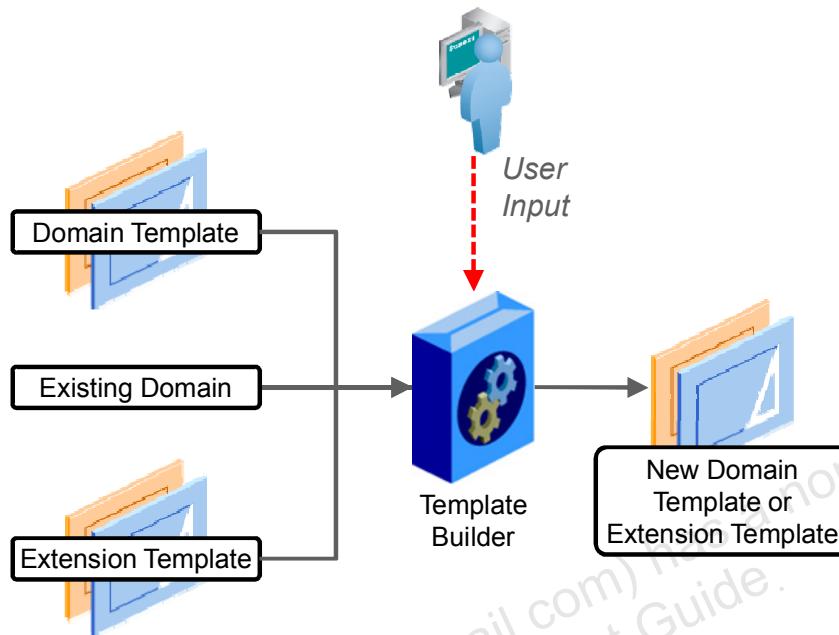
Custom domain templates allow you to easily duplicate a domain's configuration without manually configuring everything again.

- Some example uses of custom templates:
 - Definition and propagation of a standard development domain
 - Distribution of an application that runs on the domain
 - Easily move a domain from one environment to another



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Template Builder

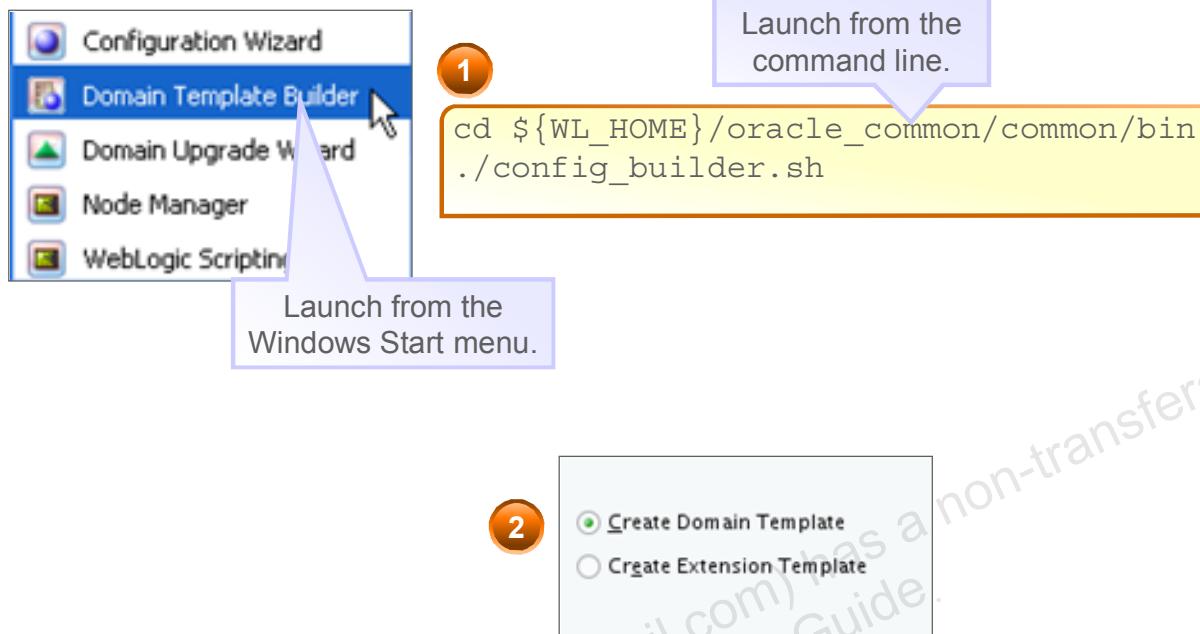


ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The Domain Template Builder enables you to create reusable custom domain templates and extension templates from existing domains or from other templates.

Creating a Domain Template

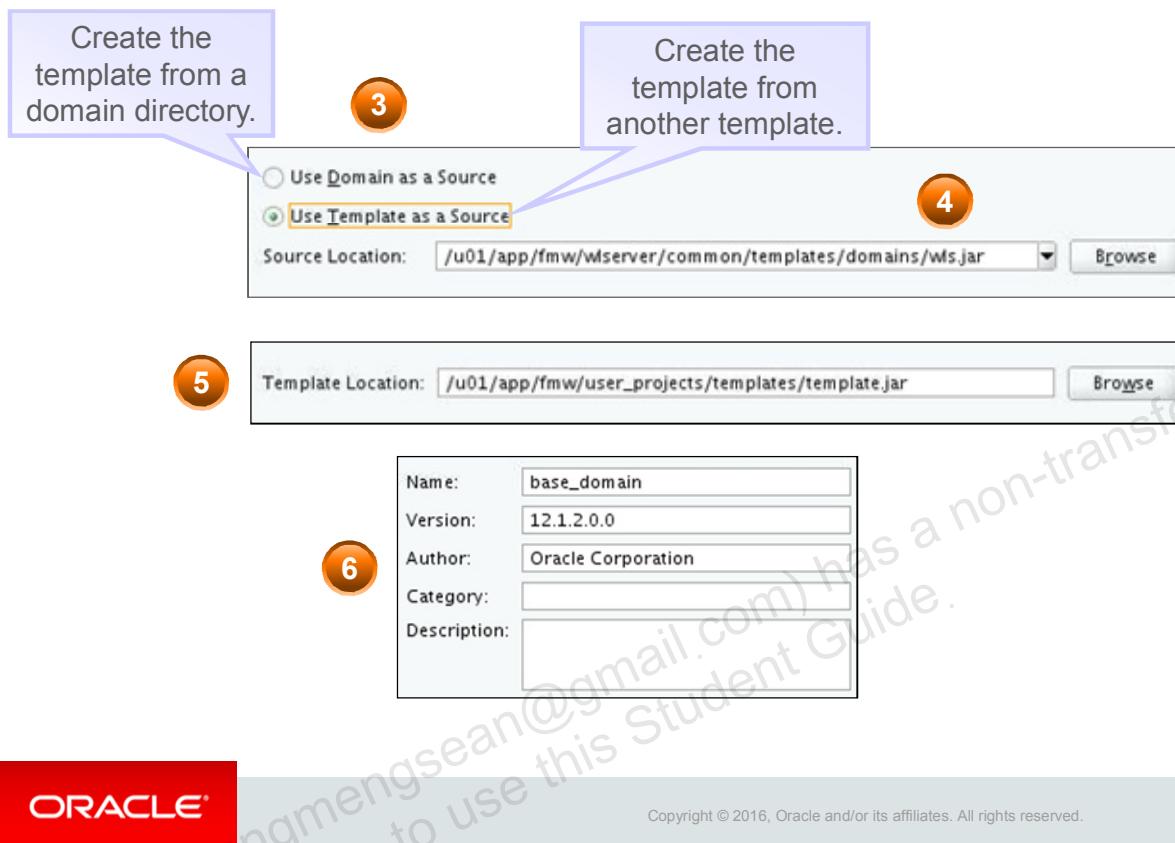


ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

1. Launch the Domain Template Builder tool.
2. To create a full domain template, select the Create Domain Template option and click Next.

Selecting the Template Domain Source



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

3. Choose from the following options: “Use Domain as a Source” and “Use Template as a Source.”
4. Select the domain or domain template from which you want to create a domain template. The templates to choose from are located in the directory specified in the Template Location field. If you want to change the directory, click Browse, and then navigate to the appropriate directory or enter the path manually.
5. Enter the template JAR name and location, and click Next.
6. Enter a name and descriptive metadata about the new template. If you selected a template as the source for the new template, information about the selected template is displayed. Review the information, change it to suit the requirements of your domain, and click Next.

Configuring Applications in a Template

7

The screenshot shows the Oracle Template Builder interface. A callout bubble points to the 'Applications' section of the configuration page. The configuration details are as follows:

Name:	AuctionLib#2.0@1.0
Location:	/practices/part2/apps/solution/AuctionDbLib.war
Internal Path:	\$APPLICATION DIRECTORY\$/AuctionDbLib.war

Below these fields is a section titled 'Applications' containing three items, all of which have checkmarks:

- AuctionLib#2.0@1.0
- AuctionWebAppSec.war

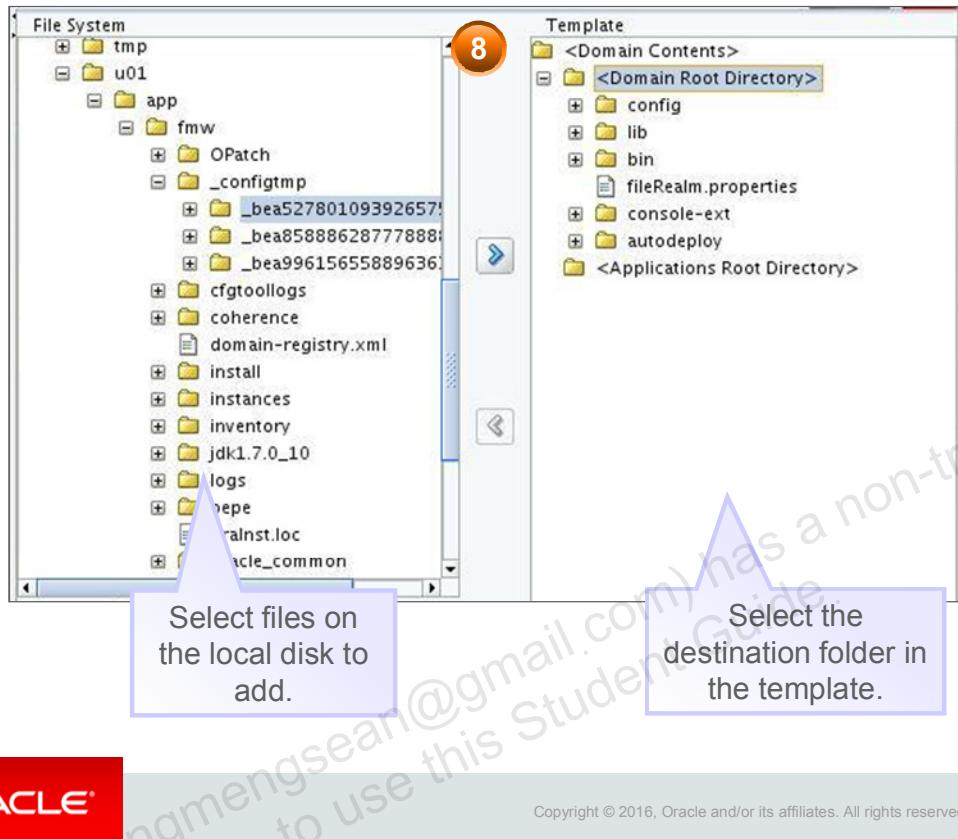
Template Builder lists applications
that are part of the domain or
template.

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

7. If there are any applications within the source domain or template, the Applications page is displayed. The Applications page enables you to review and remove files that are included in the template. You select the applications you want to include in the finished template, and click Next.

Adding Files to the Template

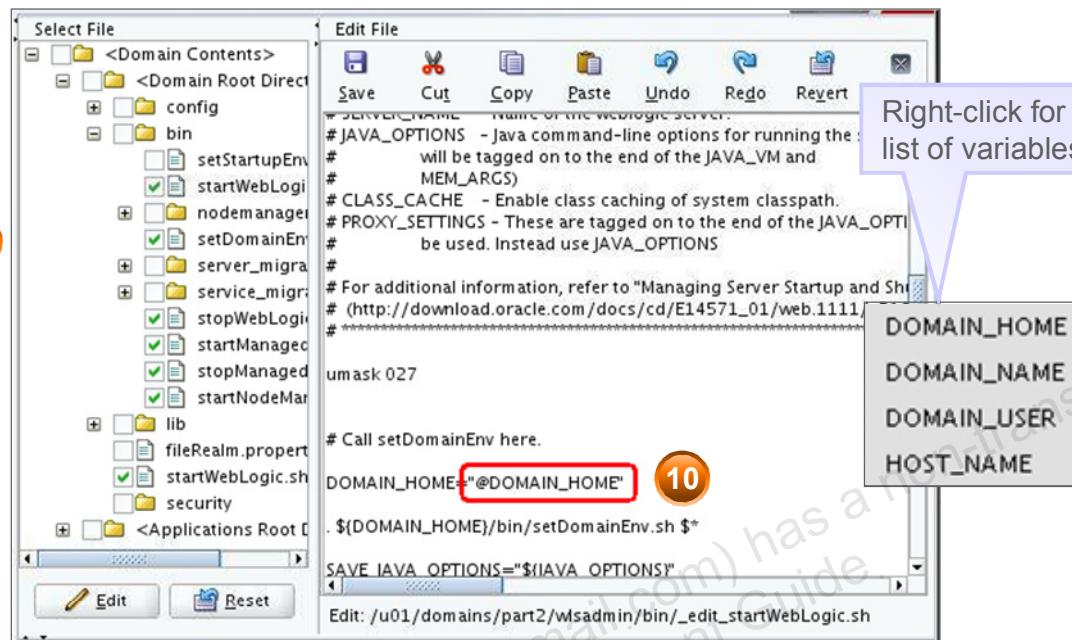


8. The Domain Content page enables you to review, add, or remove files in the template. Select a file or folder in the File System view pane, select the destination folder in the Template view pane, and then click the arrow button. When finished, click Next.

By default, the Domain Template Builder includes files from the domain or template that you specified as the source for the new template:

- If you selected an existing template as the source for the new template, all the files from the source template are automatically included. If the existing template defines a separate applications directory, the applications in the template are listed under the Applications Root Directory in the Current Template View pane.
- If you selected a domain as the source for your new template, the following files and directories are included by default for your domain:
 - All the files in the root directory with the following extensions: .cmd, .sh, .xml, .properties, and .ini
 - Any file with the extension .pem defined in the secure sockets layer (SSL) configuration
 - /bin and /lib directories
 - All the files in /security that are not created automatically during domain creation

Preparing Scripts and Files with Variables



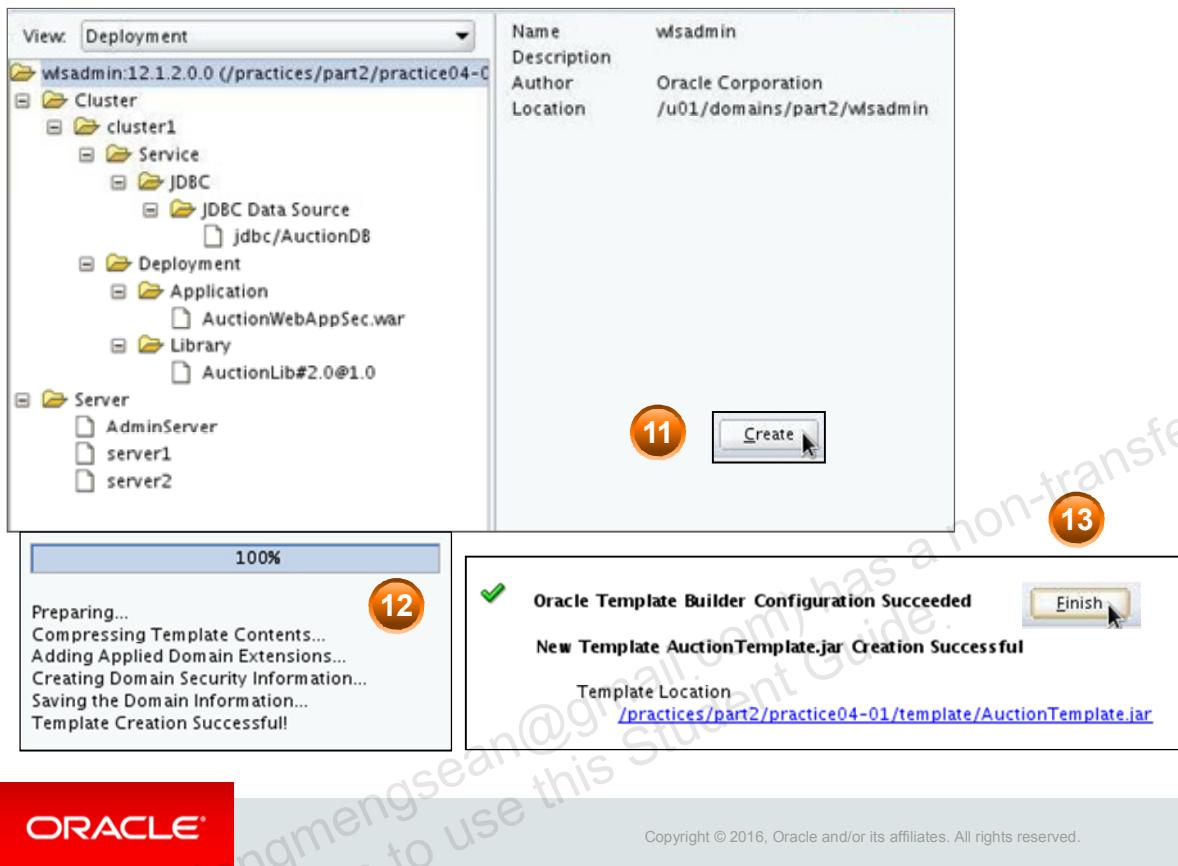
ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

9. Use the check boxes in the left pane to select the template's files that require variable replacement by the Configuration Wizard. The check boxes for files that were automatically updated by the Domain Template Builder will be already selected. To update a file and insert replacement variables in it, either double-click the file, or select the file and click Edit.
10. In the editor pane on the right, insert variables as required. For convenience, right-click and select from the list of common replacement variables. After you finish editing the file, click Save. After all the editing is complete, click Next.

Note: This is only required for custom files. Standard domain files are automatically updated by Template Builder.

Review Configuration and Create Template



11. Review the configuration summary of your new template and click Create.
12. Wait for the template to get created and click Next.
13. See the template creation status and click Finish to close the Template Builder.

SQL Scripts and LDAP Data

The Template Builder does not perform any processing specific to SQL scripts:

- SQL scripts must be included as regular files during the template creation process.
- SQL scripts are not run by the Configuration Wizard and must be run manually after domain creation.

The Template Builder does not store LDAP data in the template:

- Users, groups, and roles are not stored.
- Export LDAP data into XML files before template creation and add those files to the template.

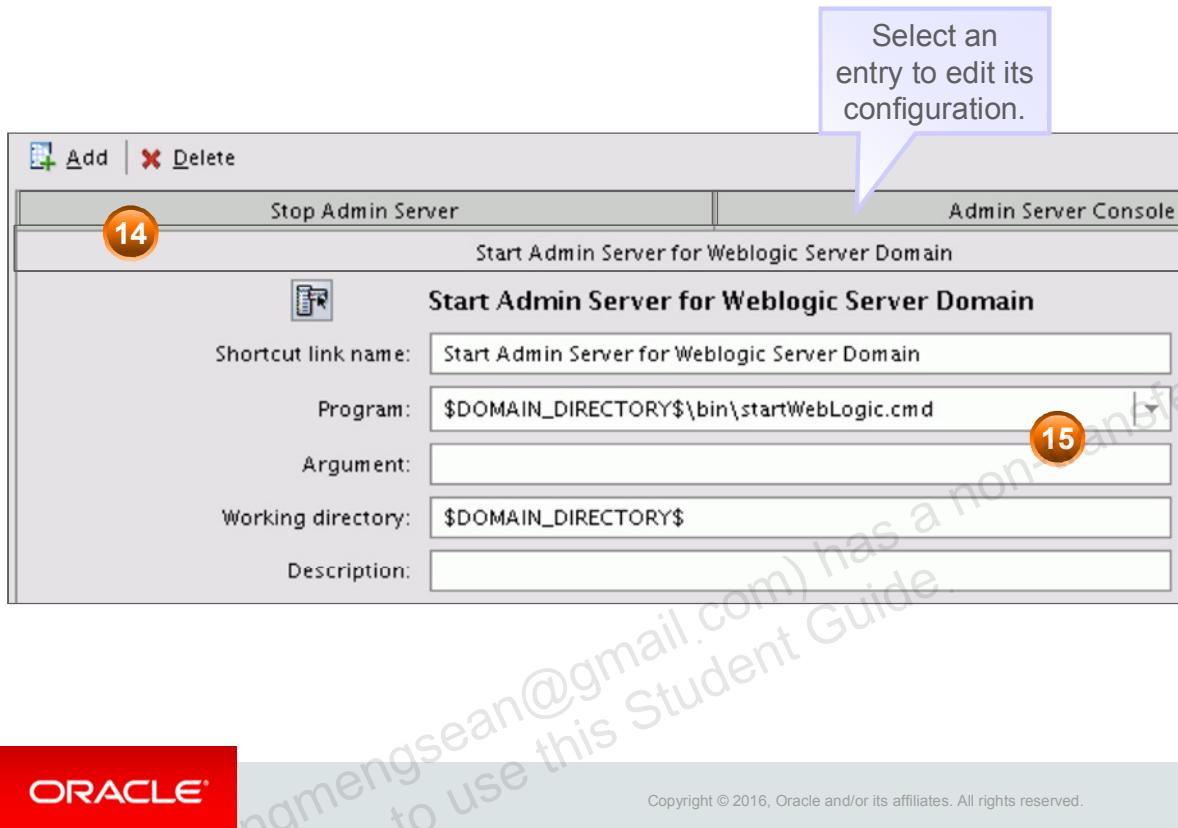


Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Oracle products that are built on WebLogic and require database creation rely on Oracle's Resource Creation Utility (RCU) to do so.

Note: The Template Builder and Configuration Wizard automatically include the standard WebLogic groups and administrative user in the template and domains created from it.

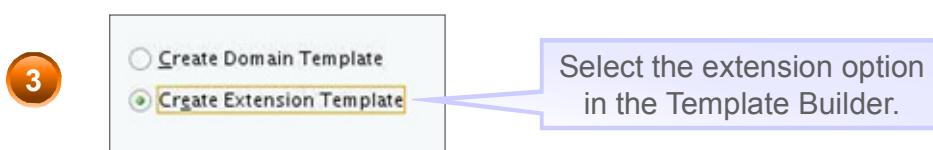
Creating Windows Start Menu Entries



14. Click Add to create a new Windows Start menu entry for this domain template. Multiple Start menu entries are shown as tabs at the top of the page. If you selected a template as the source for your custom template, any existing entries from the selected template are displayed. Review these entries and modify them if necessary.
15. After clicking a tab, enter the details of the shortcut to be placed in the Start menu. Use script replacement variables as necessary. These variables will be later replaced by the Configuration Wizard, similar to other server start scripts. When finished, click Next.

Creating an Extension Template

- 1 Perform any prerequisite tasks, such as exporting LDAP principals and including the resulting XML files in the template source.
- 2 Start the Template Builder.



Create an extension template from a domain directory.

Create an extension template from another template.

Select the extension option in the Template Builder.

- 3
- 4 Select the template source.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The process for creating an extension template is the same for creating a complete domain template, except for selecting the Create Extension Template option as shown in this slide.

Creating an Extension Template

- 5 Enter template information.
- 6 Select the applications in source to keep/omit.
- 7 Add any custom files and scripts.
- 8 Add replacement variables to any custom files.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Using a Custom Template with the Configuration Wizard

1

```
cd ${WL_HOME}/oracle_common/common/bin  
./config.sh
```

Launch from the command line.

2

What do you want to do?

- Create a new domain
 Update an existing domain

Domain Location: /u01/domains/NewAuctionDomain

3

Create Domain Using Custom Template:

Template location: /practices/part2/practice04-01/template/AuctionTemplate.jar

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Using a custom domain template is the same as using a prebuilt domain template that comes with WebLogic Server. There are only slight differences.

1. Launch the Configuration Wizard in the WebLogic common script folder.
2. Indicate the type of domain you want to create and enter the location and name of your new domain.
3. Select Create Domain Using Custom Template and enter or browse to the path and file name of your custom template JAR file.

Using a Custom Template with the Configuration Wizard

4

Domain name: NewAuctionDomain
Domain location: /u01/domains
Application location: /u01/domains/NewAuctionDomain/applications

5

Name: weblogic
Password: Confirm Password:

6

Domain Mode
 Development
Utilize boot.properties for username and password, and poll for applications to deploy. Sun JDK recommended for better startup performance during iterative development.
 Production
Require the entry of a username and password, and do not poll for applications to deploy. WebLogic JRockit JDK recommended for better runtime performance and management.
JDK
 Oracle HotSpot 1.7.0_10 /u01/app/fmw/jdk1.7.0_10
 Other JDK Location:

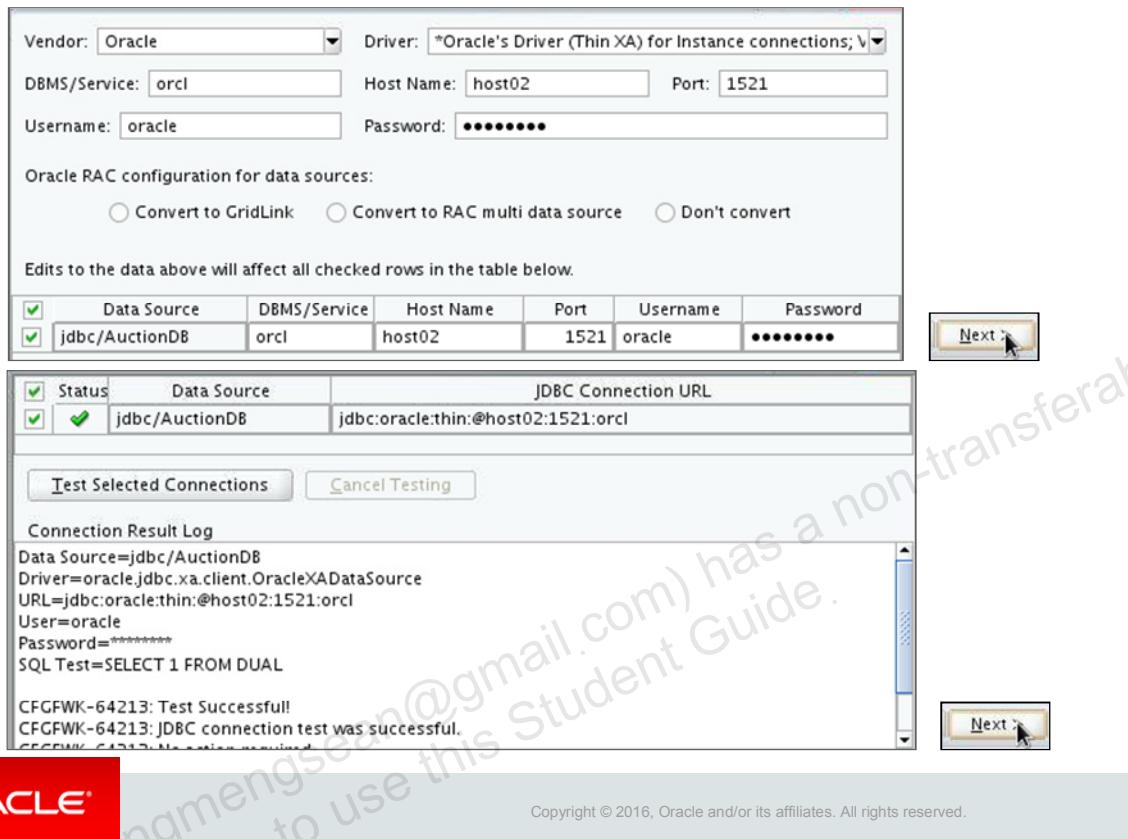


Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

4. Enter the location to place the deployed applications that are part of the template.
5. Enter the login name and password credentials for the domain's administrative account.
6. Select the Domain Mode of Development or Production, and the JDK to use for the domain.

Using a Custom Template with the Configuration Wizard

7

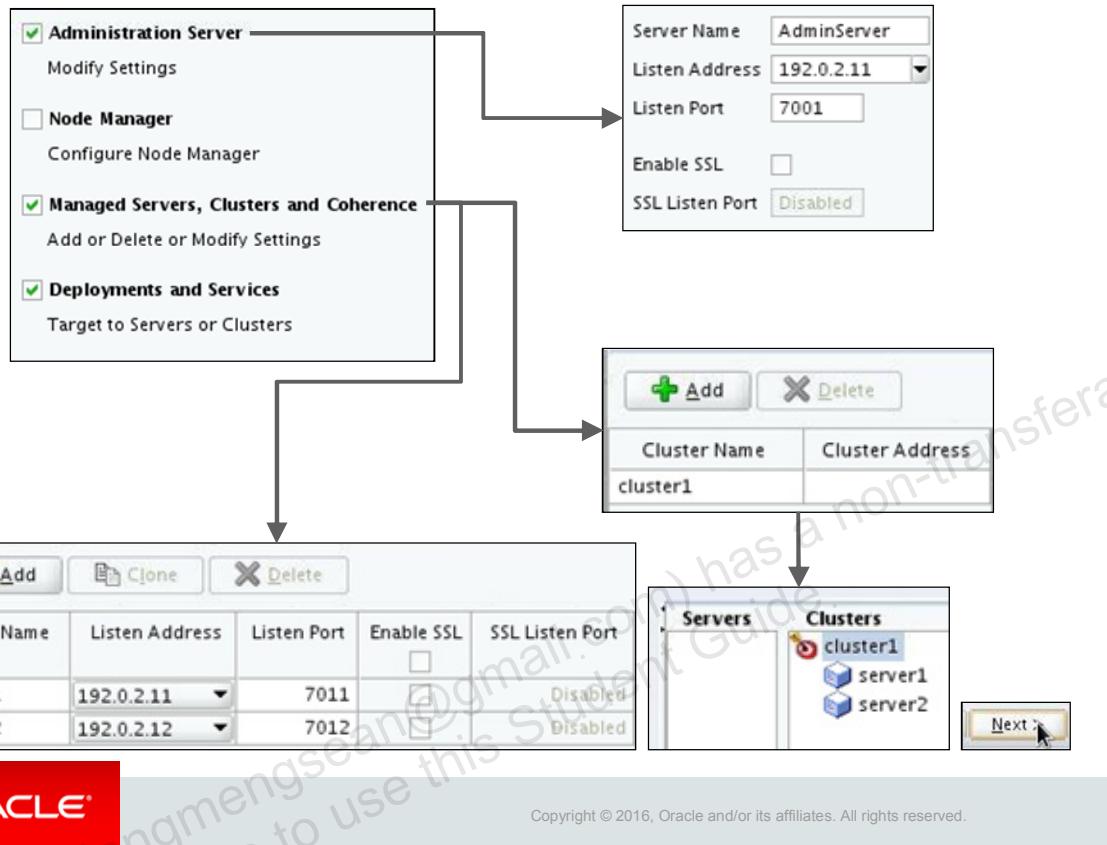


8

7. Either leave the settings at the default values set within the template or select the check box next to any JDBC data sources configured in the template to override settings. You will most likely need to modify the database host, service, port, and password settings to match the target environment.
8. Test your JDBC data source settings to ensure they are configured correctly.

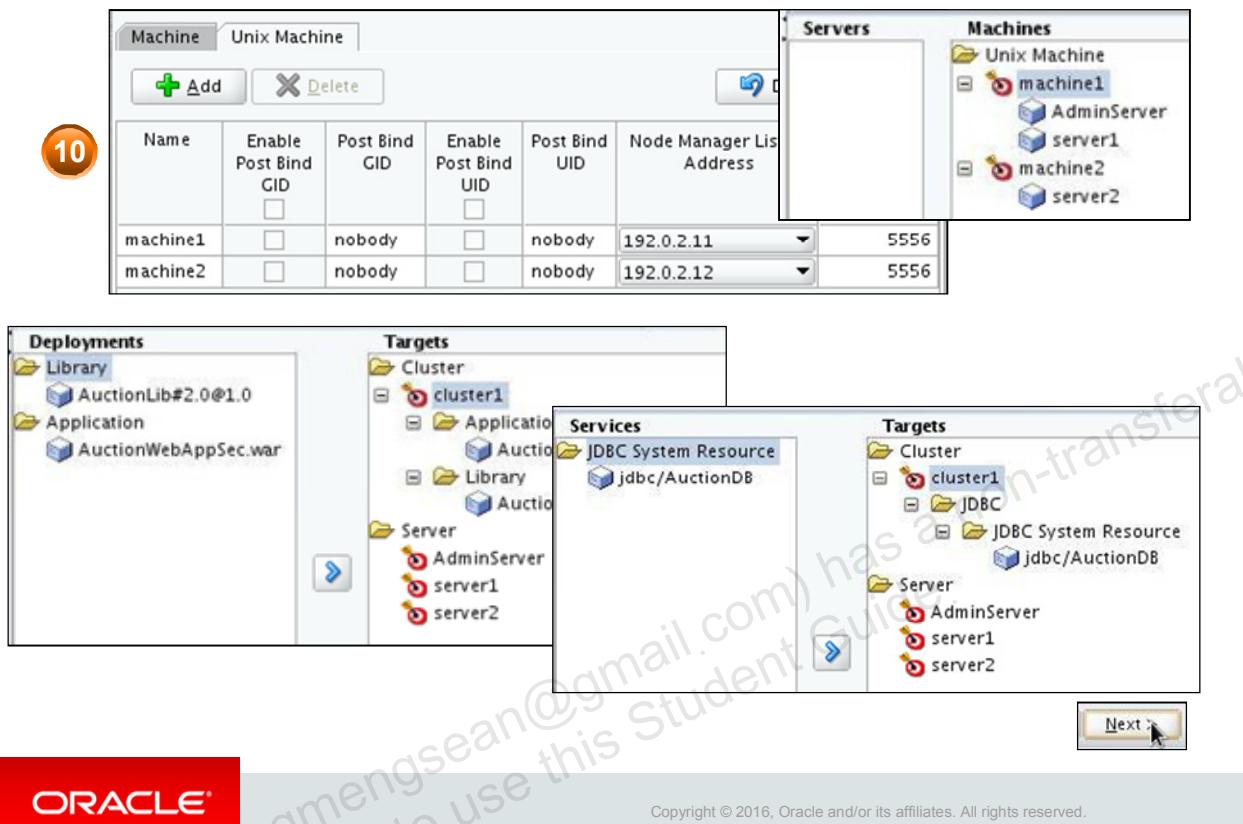
Using a Custom Template with the Configuration Wizard

9



9. Select any optional systems that you want to configure, such as the Administration Server or Managed Server settings. You will most likely need to modify the listen address and ports for your servers to match your target environment.

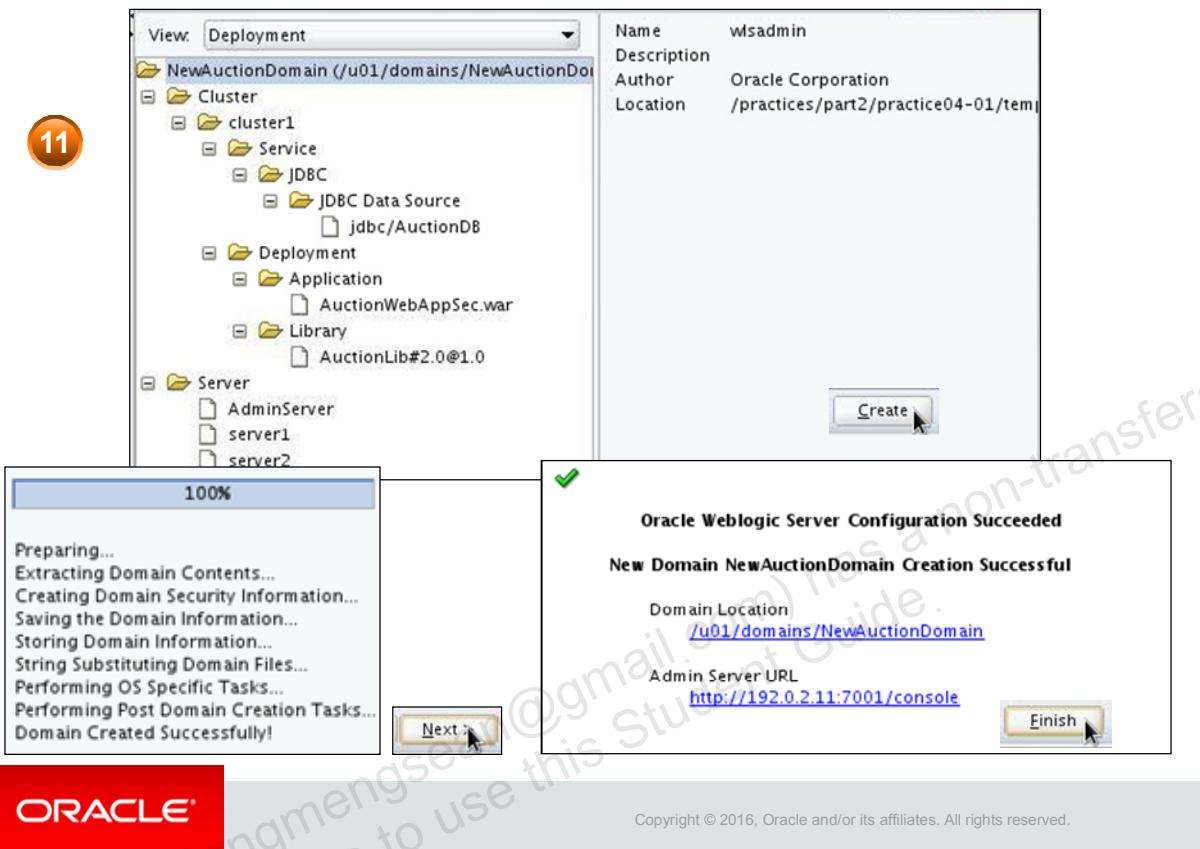
Using a Custom Template with the Configuration Wizard



10. Continue setting optional configuration values. All default values match the settings of the original template, such as machines, clusters, and server and resource assignments.

Using a Custom Template with the Configuration Wizard

11



11. Review the configuration summary and create your custom template-based domain.

Post Domain Creation Tasks

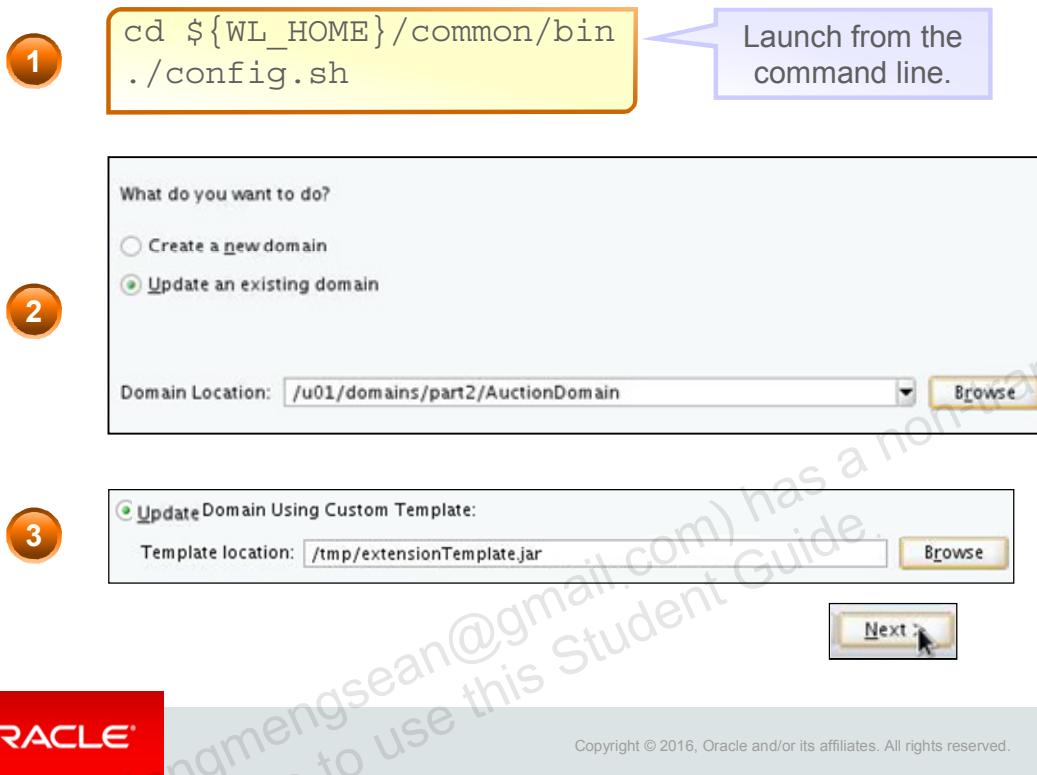
The Template Builder does not include all processing that a domain may require:

- Import LDAP data from XML files or run scripts to provision identities if using a third-party LDAP product
- Run commands to pack and unpack domain contents for managed servers on remote machines
- Run scripts to create database artifacts



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Using a Custom Extension Template with the Configuration Wizard



Using a custom domain extension template is the same as using a custom domain template. There are only slight differences.

1. Launch the Configuration Wizard in the WebLogic common script folder.
2. Select *Update an existing Domain*, which allows you to extend an existing domain, and enter the location and name of your domain.
3. Select *Update Domain Using Custom Template* and enter or browse to the path and file name of your custom template JAR file.

Using a Custom Extension Template with the Configuration Wizard

- 4 Select the location to place applications found in the template.
- 5 Verify or configure data source settings.
- 6 Optionally change advanced configuration settings.
- 7 Verify configuration summary and processing



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

- Administration server and Node Manager configuration settings cannot be changed when using an extension template.
- Advanced configuration settings that are available depend on what resources are included in the extension template.
- If you select Managed Server, Clusters, and Coherence, you can configure settings related to these resources as needed.
- Resources you can configure using extension templates:
 - Managed servers, clusters, and assigning servers to a cluster
 - HTTP proxy applications
 - Coherence clusters
 - System components (only if that system component is installed, such as OHS)
 - Deployment and services targeting
 - JMS file stores

Summary

In this lesson, you should have learned how to:

- Create a domain template using the Template Builder
- Create an extension template using the Template Builder
- Create a domain using a template
- Extend a domain using a template



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Practice 4-1 Overview: Creating and Using a Custom Domain Template

This practice covers the following topics:

- Using the Domain Template Builder to create a template from an existing domain
- Adding custom files and SQL scripts to a template
- Configuring variable replacement in template files
- Creating a new domain from a custom template
- Using the Auction application on the newly created domain



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

5

WebLogic Server Startup and Crash Recovery

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to configure the Node Manager to start:

- Failed servers
- On system boot



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Agenda

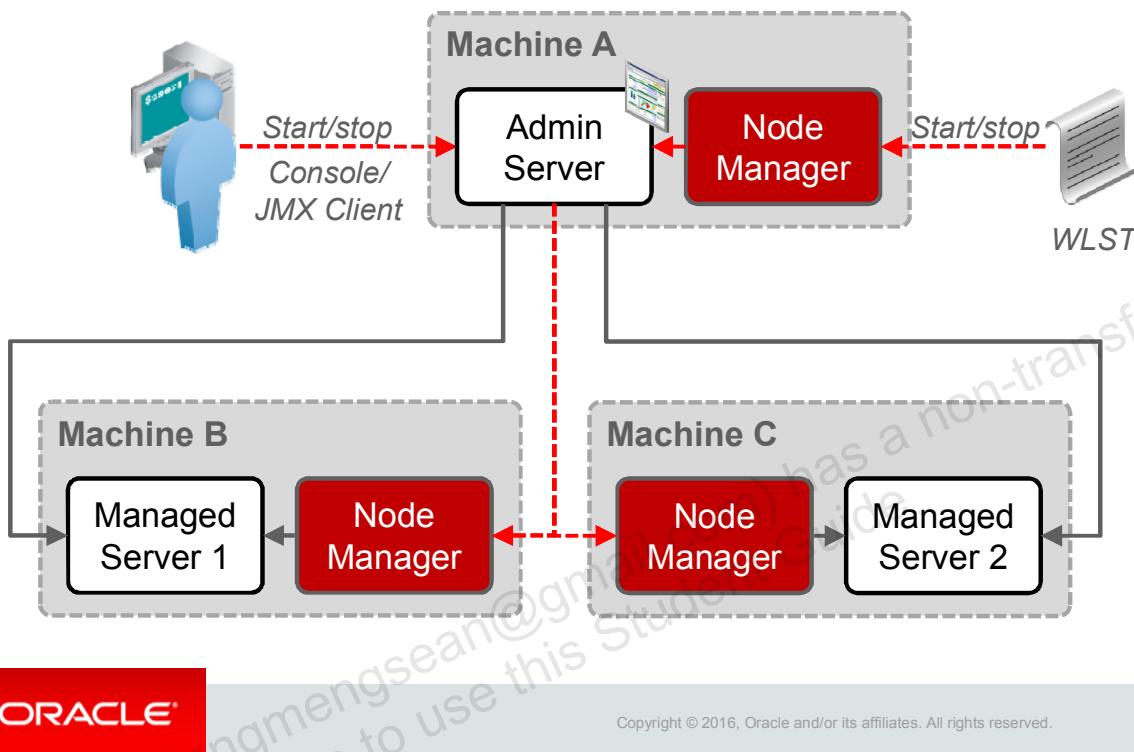
- Node Manager Review
 - Node Manager Architecture
 - Node Manager Default Behavior
 - Configure Java-Based Node Manager
- Configure the Node Manager to Start on System Boot
- Server Automatic Restart



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Node Manager Architecture

Each Node Manager monitors and restarts servers it has started.



The graphic in the slide illustrates the relationship between a Node Manager, its clients, and the server instances that it controls. You can access the Node Manager using the following clients:

- **Administration server:** From the administration console, select Environments > Machines > Configuration > Node Manager page.
- **JMX utilities**
- **WebLogic Scripting Tool (WLST) commands and scripts:** WLST offline serves as the Node Manager command-line interface that can run in the absence of a running administration server. You can use the WLST commands to start, stop, and monitor a server instance without connecting to an administration server. Starting the administration server is the main purpose of the stand-alone client. However, you can also use it to:
 - Stop a server instance that was started by a Node Manager
 - Start a managed server
 - Access the contents of a Node Manager log file
 - Obtain the server status for a server that was started with a Node Manager
 - Retrieve the contents of the server output log

Node Manager Default Behavior

- After WebLogic is installed and a domain is created, the Node Manager is “ready-to-run”.
- By default, the following behaviors are configured:
 - The Node Manager can start servers using WLST.
 - The administration server uses Node Managers to start managed servers.
 - The Node Manager monitors all the servers that it started.
 - The automatic restart of all servers is enabled.

This is initiated by the administration console or WLST.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The Node Manager is ready to run after WebLogic Server is installed, a domain is created, and you use the demonstration secure sockets layer (SSL) configuration. The Node Manager restarts the server instances that it killed or that terminated unexpectedly.

Configure Java-Based Node Manager

- It is recommended that you configure the Node Manager to run as an operating system service.
- The configuration tasks for the Java-based Node Manager include:
 - Reconfiguring the startup service for a Windows installation
 - Running the Node Manager as a daemon process for UNIX systems
 - Configuring Java-based Node Manager security
 - Reviewing `nodemanager.properties`
 - Configuring the Node Manager on multiple machines



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

- `nodemanager.properties`: This is the configuration file that is used by the Java-based version of Node Manager. This file is located in `WL_HOME/common/nodemanager`.
- `nodemanager.domains`: This file contains mappings between the names of the domains managed by the Node Manager and their corresponding directories. This file is located in `WL_HOME/common/nodemanager`.
- `nm_data.properties`: This file stores the encryption data that the Node Manager uses as a symmetric encryption key. The data is stored in an encrypted form. This file is located in `WL_HOME/common/nodemanager`.

Agenda

- Node Manager Review
- Configure the Node Manager to Start on System Boot
 - Windows
 - Linux
 - Solaris
- Server Automatic Restart



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Starting the Node Manager at System Startup

- It is recommended that you run the Node Manager as:
 - A Windows service on Windows platforms
 - A daemon process on UNIX platforms
- Running the Node Manager during system startup allows it to restart automatically when the system is rebooted.
- You configure the Node Manager to start at boot time as:
 - A Windows service
 - A UNIX daemon process



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Configuring the Node Manager as a Windows Service

1. Delete the Node Manager service by using `uninstallNodeMgrSvc.cmd`.
If it exists already.
2. Edit `installNodeMgrSvc.cmd` to specify the Node Manager's listen address and listen port.
3. Run `installNodeMgrSvc.cmd` to reinstall the Node Manager as a service, listening on the updated address and port.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The `$WL_HOME/server/bin` directory (where `WL_HOME` is the top-level directory for the Oracle WebLogic Server installation) contains `uninstallNodeMgrSvc.cmd`, a script for uninstalling the Node Manager service, and `installNodeMgrSvc.cmd`, a script for installing the Node Manager as a service.

1. Delete the Node Manager service by using `uninstallNodeMgrSvc.cmd`.
2. Edit `installNodeMgrSvc.cmd` to specify Node Manager's listen address and listen port.

Note: By default, the service installer configures the Node Manager to run on `localhost:5556`. You edit the installation file to change the host address and port number. This slide refers to this reconfiguration.

Make the same edits to `uninstallNodeMgrSvc.cmd` as you make to `installNodeMgrSvc.cmd`, so that you can successfully uninstall the service in the future.

3. Run `installNodeMgrSvc.cmd` to reinstall the Node Manager as a service, listening on the updated address and port.

Configuring the Node Manager on UNIX Systems

UNIX and Linux operating system provide two different utilities for running services:

Utility	Description
inetd	Internet service daemon that manages Internet-based services.
xinetd	Extended Internet daemon that managers Internet-based services. Xinetd offers access control mechanisms including access control lists, logging capabilities, service availability by time, and limits on number of servers to start.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

On UNIX, the Node Manager must be configured to run as a daemon manually.

Configuring the Node Manager as an `inetd` Service

```
#!/bin/sh
# nodemgr Oracle Weblogic NodeManager service
# chkconfig: 345 85 15
# description: Oracle Weblogic NodeManager service
# The script needs to be saved as /etc/init.d/nodemgr and
# then issue chkconfig --add nodemgr as root

### BEGIN INIT INFO
# Provides: nodemgr
# Required-Start: $network $local_fs
# Required-Stop:
# Should-Start:
# Should-Stop:
# Default-Start: 3 4 5
# Default-Stop: 0 1 2 6
# Short-Description: Oracle Weblogic NodeManager service.
# Description: Starts and stops Oracle Weblogic NodeManager.
### END INIT INFO
```

- Start this service for run levels 3, 4, and 5.
- The service start priority is 85.
- The service stop priority is 15.

Name of the service

Services this service depends on

Run levels to start this service

Run levels to stop this service

/etc/init.d/nodemgrsvc



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Configuring the Node Manager as an `inetd` Service

```
. /etc/rc.d/init.d/functions                                Source initd internal functions

export APP="/u01/app"
export MW_HOME="/u01/app/fmw"
export JAVA_HOME="$APP/jdk1.7.0_15"
DAEMON_USER="oracle"
PROCESS_STRING="^.*$/u01/app/fmw/*weblogic.NodeManager.*"

source $MW_HOME/wlserver/server/bin/setWLSEnv.sh > /dev/null
export NODEMGR_HOME="/u01/nodemanager"
NodeManagerLockFile="$NODEMGR_HOME/nodemanager.log.lck"

PROGRAM="$MW_HOME/wlserver/server/bin/startNodeManager.sh"
SERVICE_NAME=`basename $0`
LOCKFILE="/var/lock/subsys/$SERVICE_NAME"
RETVAL=0
```

Source initd internal functions

Set variables used by the script

/etc/init.d/nodemgrsvc



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Configuring the Node Manager as an `inetd` Service

```
start() { <script to start Node Manager> }
stop() {<script to stop Node Manager> }
restart() {<script to restart Node Manager> }

case "$1" in
start)
    start
    ;;
stop)
    stop
    ;;
restart|force-reload|reload)
    restart
    ;;
...
esac

exit $RETVAL
```

Call above start function

Call above stop function

Call above restart function

/etc/init.d/nodemgrsvc



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Registering and Managing `inetd` Services

- You register a service with `inetd` using the `chkconfig` command as the `root` user.

```
$ chkconfig --add nodemgr
```

Command Line

- The following commands are used to manage the lifecycle of `inetd` services:

```
$ service nodemgr start  
$ service nodemgr stop  
$ service nodemgr restart  
$ service nodemgr status
```

Command Line



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Configuring the Node Manager as an `xinetd` Service

- When configuring the Node Manager to run as an `xinetd` service, consider the following:
 - Ensure that `NodeManagerHome` and other system properties are defined.
 - If `xinetd` is configured with `libwrap`, you should add the `NOLIBWRAP` flag.
 - Ensure that the `hosts.deny` and `hosts.allow` files are configured correctly.
 - Depending on your network environment, additional configuration may be necessary.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

NOLIBWRAP from `xinetd` man page: This disables internal calling of the `tcp-wrap` library to determine access to the service. This may be needed in order to use `libwrap` functionality not available to long-running processes such as `xinetd`; in this case, the `tcpd` program can be called explicitly (see also the `NAMEINARGS` flag). For RPC services using TCP transport, this flag is automatically turned on, because `xinetd` cannot get remote host address information for the RPC port.

`hosts.deny` and `hosts.allow` files: These provide access control lists of remote hosts that are granted or denied access to daemon processes running on this machine. The search for access stops at the first search in the following order:

- Access is granted when a client and daemon pair matches in `hosts.allow`.
- Access is denied when a client and daemon pair matches in `hosts.deny`.
- Access is granted if there are no matches in either file.

Configuring the Node Manager as an `xinetd` Service

```
# default: off
# description:nodemanager as a service
service nodemgrsvc
{
    type = UNLISTED
    disable = no
    socket_type = stream
    protocol = tcp
    wait = yes
    user = <username>
    port = 5556
    flags = NOLIBWRAP
    log_on_success += DURATION HOST USERID
    server = <path-to-java>/java
    env = CLASSPATH=<cp> LD_LIBRARY_PATH=<ldpath>
    server_args = -client -DNodeManagerHome=<NMHome> <java options>
                  <nodemanager options> weblogic.NodeManager -v
}
```

/etc/xinetd.d/nodemgrsvc



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This slide shows an example of how the Node Manager can be configured within `xinetd`. The following commands are used to manage the lifecycle of `xinetd` and its services (run as root):

- `service xinetd start`
- `service xinetd stop`
- `service xinetd restart`
- `service xinetd status`

Solaris

Solaris provides an alternative to xinetd and inetd service management. In addition to these services, it also provides the Service Management Facility (SMF).

- SMF provides:
 - Consistent interface
 - Dependency ordering
 - Delegation of tasks to non-root users
 - Parallel starting of services
 - Automatic service restart after failure



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Solaris offers SMF in addition to xinetd. Some say it is a more modern and complete offering. SMF provides the following:

- **Consistent interface:** SMF uses manifest files to manage services, rather than using specialized scripts and configuration files for each service.
- **Dependency order:** SMF allows you to make a service dependent on another service so services required by other services start first.
- **Delegation of tasks:** Allows services to run as users other than root. This protects the system by using the privileges of a user with more restrictions than the root user.
- **Parallel start:** Services are started in parallel, which speeds up boot time by running services simultaneously.
- **Automatic restart:** Works with the Solaris Fault Manager (SFM) to enable software recovery in the event of a failure.

Example SMF Manifest File

```
<?xml version='1.0'?>
<!DOCTYPE service_bundle SYSTEM
  '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>
<service_bundle type='manifest' name='nodemanager'>
<service name='application/management/nodemanager/oracle'
  type='service' version='1'>
  <single_instance/>
  <dependency
    name='multi-user-server' grouping='require_any'
    restart_on='error' type='service'>
    <service_fmri value='svc:/milestone/multi-user-server:default' />
  </dependency>
  <exec_method type='method' name='start'
    exec='/u01/domains/wlsadmin/bin/startNodeManager.sh'
    timeout_seconds='120' >
    <method_context>
      <method_credential user='oracle' group='oinstall' />
    </method_context>
  </exec_method>
```

Start as oracle user

nodemanager.xml



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The manifest file specifies the domain's `startNodeManager.sh` script as the script to run when the service is starting. This script is included with WebLogic.

Example SMF Manifest File

```
<exec_method type='method' name='stop'  
    exec='/u01/domains/wlsadmin/bin/stopNodeManager.sh'  
    timeout_seconds='120'>  
    <method_context>  
        <method_credential user='oracle' group='oinstall'/>  
    </method_context>  
</exec_method>  
<property_group name='start' type='method'>  
    <propval name='action_authorization' type='astring'  
        value='solaris.smf.manage.nodemanager/oracle'/>  
    <propval name='modify_authorization' type='astring'  
        value='solaris.smf.manage.nodemanager/oracle'/>  
    <propval name='value_authorization' type='astring'  
        value='solaris.smf.manage.nodemanager/oracle'/>  
</property_group>
```

Stop as oracle user

Authorize oracle to start

nodemanager.xml



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The manifest file also specifies the domain's `stopNodeManager.sh` script as the script to run when the service is stopping. This script must be created by you because it is not included with Weblogic by default. All the script should do is kill the Node Manager process.

Example SMF Manifest File

```
<property_group name='stop' type='method'>
    <propval name='action_authorization' type='astring'
        value='solaris.smf.manage.nodemanager/oracle'/>
    <propval name='modify_authorization' type='astring'
        value='solaris.smf.manage.nodemanager/oracle'/>
    <propval name='value_authorization' type='astring'
        value='solaris.smf.manage.nodemanager/oracle'/>
</property_group>
<property_group name='general' type='method'>
    <propval name='action_authorization' type='astring'
        value='solaris.smf.manage.nodemanager/oracle'/>
    <propval name='modify_authorization' type='astring'
        value='solaris.smf.manage.nodemanager/oracle'/>
    <propval name='value_authorization' type='astring'
        value='solaris.smf.manage.nodemanager/oracle'/>
</property_group>
.
.
</service>
</service_bundle>
```

Authorize oracle to stop

Authorize oracle for general commands

nodemanager.xml



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Set Up and Start Node Manager with SMF

Validate manifest:

```
svccfg validate nodemanager.xml
```

Command Line

Give oracle user authorization (as root user):

```
solaris.smf.manage.nodemanager/oracle:::NodeManager Management::
```

/etc/security/auth_attr

```
usermod -A solaris.smf.manage.nodemanager/oracle oracle
```

Command Line

Import the manifest (as root user):

```
svccfg import nodemanager.xml
```

Command Line

Enable or disable the service as the oracle user:

```
svcadm enable application/management/nodemanager/oracle
```

```
svcadm disable application/management/nodemanager/oracle
```

Command Line

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

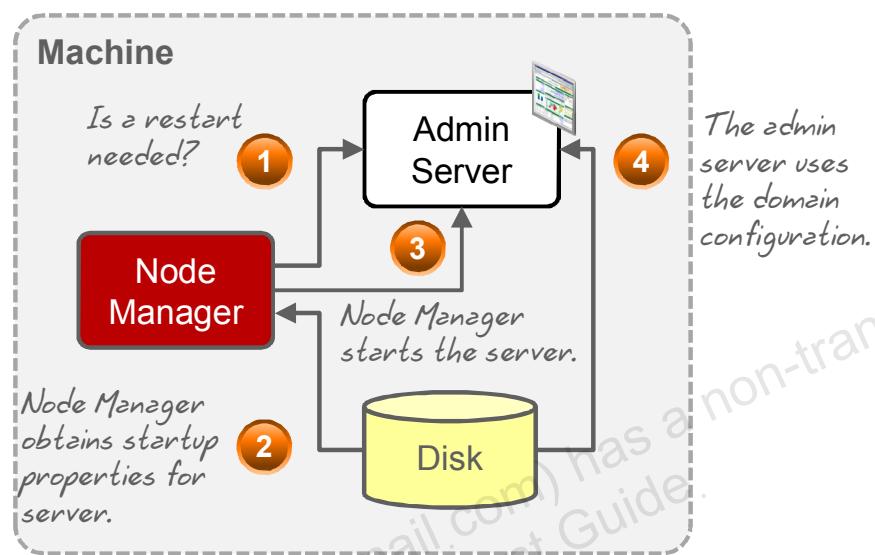
Agenda

- Node Manager Review
- Configure the Node Manager to Start on System Boot
- Server Automatic Restart
 - Administration Server
 - Managed Server
 - Crash Recovery
 - Configuring Node Manager Restart Parameters



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

How the Node Manager Restarts an Administration Server

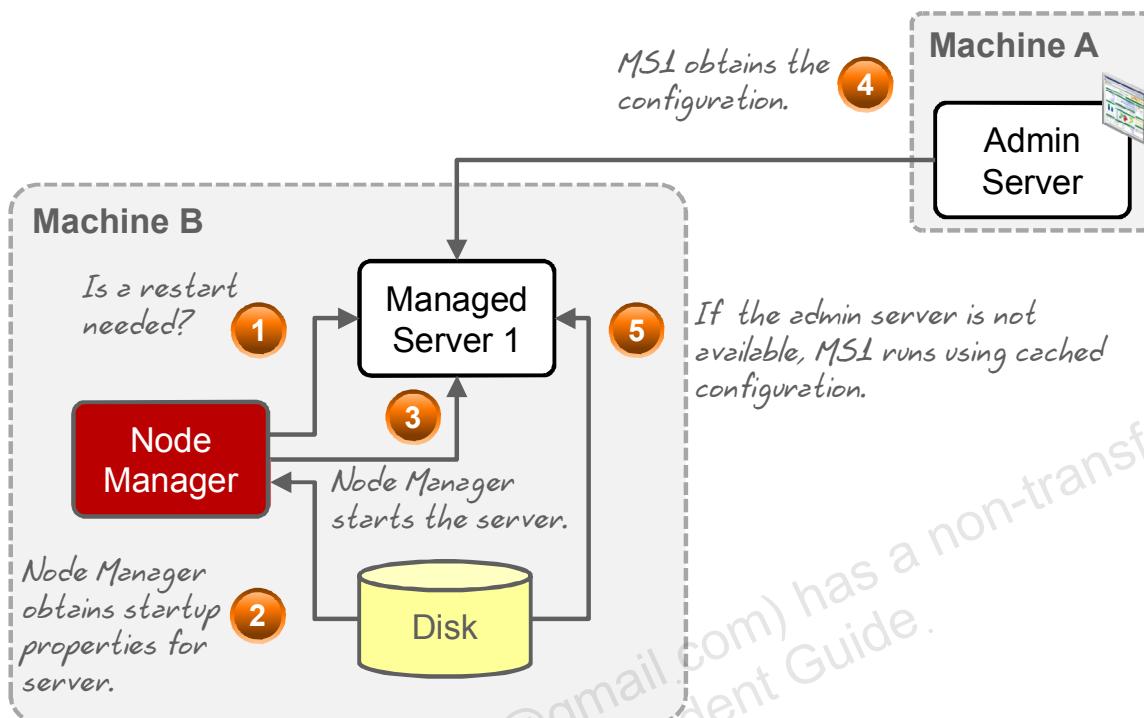


ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

1. The Node Manager determines from the administration server process exit code that it requires a restart.
2. The Node Manager obtains the username and password for starting the administration server from the `boot.properties` file, and the server startup properties from the `server/security/startup.properties` file. These server-specific files are located in the `server` directory for the administration server.
3. The Node Manager starts the administration server.
4. The administration server reads its configuration data and starts up.

How the Node Manager Restarts a Managed Server



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

1. The Node Manager determines from Managed Server 1's last known state that it requires a restart.
2. The Node Manager obtains the username and password for starting Managed Server 1 from the `boot.properties` file, and the server startup properties from the `startup.properties` file. These server-specific files are located in the `server` directory for Managed Server 1.
3. The Node Manager starts Managed Server 1.
Note: The Node Manager waits `RestartDelaySeconds` after a server instance fails before attempting to restart it.
4. Managed Server 1 attempts to contact the administration server to check for updates to its configuration data. If it contacts the administration server and obtains updated configuration data, it updates its local cache in the `configuration` directory.
5. If Managed Server 1 fails to contact the administration server, and if Managed Server Independence (MSI) mode is enabled, Managed Server 1 uses its locally cached configuration data.

RestartInterval and RestartMax

Property	Description	Default Value
RestartInterval	The amount of time Node Manager will spend attempting to restart a failed server	0 (unlimited)
RestartMax	The number of times the Node Manager attempts to restart a failed server within the time defined by RestartInterval	N/A (unlimited)

Example:

```
#Node manager properties
...
RestartMax=5
RestartInterval=60
```

nodemanager.properties



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Crash Recovery

- Scenario:
 - Node Managers are configured to start as system processes.
 - Node Managers were used to start all servers in the domain.
 - All the machines experienced a power outage.

What happens by default when the machines start up again?

- a. Node Managers start and automatically restart all servers in the domain.
- b. Node Managers start and do not automatically restart all servers in the domain.
- c. Node Managers must be started manually.
- d. All servers in the domain must be started manually.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This is a trick question for people new to WebLogic. The correct answer is “b” because although Node Managers automatically restart failed servers by default, they do not restart any servers in the event of a complete server crash.

- Answer “a” is incorrect because the Node Managers do not restart servers automatically after a system crash.
- Answer “c” is incorrect because the Node Managers are configured as system processes and start with the system.
- Answer “d” is both correct and incorrect because if the term servers only refers to WebLogic servers, then by default it is correct. If this includes Node Managers, it is incorrect because they start automatically.

You set the `CrashRecoveryEnabled` property to `true` to allow the Node Manager to automatically restart all domain servers in the event of a system crash.

CrashRecoveryEnabled

Property	Description	Default Value
CrashRecoveryEnabled	Determines if the Node Manager will restart the servers under its control in the event of a machine failure	false

Example:

```
#Node manager properties  
...  
CrashRecoveryEnabled=true
```

nodemanager.properties



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This slide shows the CrashRecoveryEnabled property found in the nodemanager.properties file.

Quiz



Which two of the following are reasons to start Node Manager as a system service?

- a. So Node Manager is automatically started when the machine is started
- b. So Node Manager can monitor WebLogic servers
- c. So Node Manager can automatically restart WebLogic servers
- d. So Node Manager can kill failing WebLogic servers



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: a, c

Summary

In this lesson, you should have learned how to configure the Node Manager to start:

- Failed servers
- On system boot



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Practice 5-1 Overview: Configuring Automatic Start and Restart of a System

This practice covers the following topics:

- Setting CrashRecoveryEnabled to true in the nodemanager.properties file
- Using the Node Manager to start a domain with multiple servers
- Crashing and restarting machines to see the Node Manager start on boot and restart all servers



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

6

WebLogic Scripting Tool (WLST)

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Run commands in WLST interactive mode
- Write simple WLST scripts
- Run WLST scripts



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Agenda

- WebLogic Scripting Tool (WLST)
- Jython concepts
- WLST concepts
- Java Management eXtension (JMX) concepts
- Common WLST tasks
- Fusion Middleware (FMW) commands



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic Scripting Tool (WLST)

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



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

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

Agenda

- WebLogic Scripting Tool (WLST)
- Jython concepts
 - Variable declaration
 - Conditional and loop expressions
 - I/O commands
 - Exception handling
- WLST concepts
- Java Management eXtension (JMX) concepts
- Common WLST tasks
- Fusion Middleware (FMW) commands



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Jython

Jython is a Java implementation of the popular Python scripting language:

- Simple and clear syntax
- Indentation to structure code (white space critical)
- Interactive command mode
- Custom commands
- Integration with any existing Java libraries

```
list = ['ab', 'cd', 'ef']

if len(list) >= 3:
    for x in list:
        print x, len(x)
print 'done'
```

```
from java.util import ArrayList

list = ArrayList()
list.add('ab')
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The latest Jython release implements the same language as CPython and many of the CPython standard library modules.

Jython is an implementation of the high-level, dynamic, object-oriented language Python, seamlessly integrated with the Java platform. Jython is freely available for both commercial and noncommercial use, and is distributed with source code. Jython is complementary to Java and is especially suited for the following tasks:

- **Embedded scripting:** Java programmers can add Jython libraries to their system to allow end users to write simple or complicated scripts that add functionality to the application.
- **Interactive experimentation:** Jython provides an interactive interpreter that can be used to interact with Java packages or with running Java applications. This allows programmers to experiment and debug any Java system using Jython.
- **Rapid application development:** Python programs are typically two to ten times shorter than the equivalent Java program. This translates directly to increased programmer productivity. The seamless interaction between Python and Java enables developers to freely mix the two languages both during development and in shipping products.

Python (and therefore, Jython) uses leading white space to format structures such as conditions, loops, and functions, instead of braces ("{"}) like in Java.

Using Jython

Jython can interpret commands in two ways for administrative users:

Interactive: Supply commands one at a time from a command prompt. Enter a command in the WLST console and view the response immediately.

Batch: Provide a series of commands in a script file (.py) . You create a text file with the .py extension that contains a series of WLST commands.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Interactive mode, in which you enter a command and view the response at a command-line prompt, is useful for learning the tool, prototyping the command syntax, and verifying configuration options before building a script. Using WLST interactively is particularly useful to get immediate feedback after making a critical configuration change.

The WLST scripting shell maintains a persistent connection with an instance of Oracle WebLogic Server. WLST can also write all the commands that you enter during a WLST session to a file. You can edit this file and run it as a WLST script.

Scripts invoke a sequence of WLST commands without requiring your input, much like a shell script. Scripts contain WLST commands in a text file that by convention ends with a .py file extension—for example, `myscriptfile.py`. Several sample scripts can be found in the WLST documentation online and at `<WL_HOME>\samples\server`.

Variable Declaration

```
i=0  
  
groups = ['AllUsers', 'Employees', 'Customers', 'HR']  
  
membership = [None, 'AllUsers', 'AllUsers', 'Employees']  
  
for group in groups:  
    if membership[i] is not None:  
        print 'Group ' + group + ' is a member of ' + membership[i]  
  
    i += 1
```

A simple declaration

Two static arrays

New variable group is defined as part of this for statement.

This array member is accessed using an index variable.

Variables used in a print statement

Simple increment of a variable by 1



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Jython provides the ability to do everything with variables that are found in the most common programming languages. This slide shows only a few examples of how they are declared and how they are used.

Conditional Expressions

```
if x and y:  
    #do work  
    #do more work  
  
if x > y:  
    print 'x is greater than y'  
elif x == y:  
    print 'x is equal to y'  
else:  
    print 'x is not greater than or equal to y'  
  
if x <> y:    #do work  
if x != y:    #do work  
if x is y:    #do work  
  
result = x > y ? 'greater' : 'not greater'
```

If x is true and y is true
then do this work.

If x is greater than y,
then print this message.

Otherwise, if x is equal to y
then print this message.

Otherwise, if none of the above is
true then print this message.

If x is not equal to y, then do this.

If x is the same object as y, then do this.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This slide shows examples of conditional expressions that determine the flow of a script.

Loop Expressions

```
while x < 10:  
    x += 1  
#out of loop  
  
for i in range(10):  
    print i  
#out of loop  
  
names = ['Tom', 'Dick', 'Harry']  
for name in names:  
    print 'Name=' + name  
#out of loop  
  
while 1:  
    x += 1  
    if x % 2 != 0: continue  
    print 'Only print even numbers: ' + x  
    if x == 10: break  
#out of loop
```

Loop until x is equal to 10.

Loop until i is equal to 10 and print the value of i.

Loop through each value in the names array and print each name value.

Loop indefinitely.

Increment the value of x by 1.

Loop control: Start next iteration if x is not an even number.

Loop control: Stop loop when x is equal to 10.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This slide shows examples of different types of loops, which are sections of code that repeat until certain conditions are met.

I/O Commands

```
import sys  
  
f = open(sys.argv[1], "r")  
text = f.read()  
text = f.readlines()  
f.close()  
  
f = open(sys.argv[2], "w")  
f.write(text)  
f.writelines(text)  
print >>f, 'This text is written to the file'  
f.close()
```

Import the Jython sys module for interacting with the operating system.

Open a file for reading.

Reads x bytes (all by default).

Reads all lines of the file into a list.

Closes a file

Open a file for writing.

Another way to write to a file



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This slide shows examples of reading and writing to a file on disk. When command line arguments are passed to a Jython script, they are stored in the `sys.argv` array variable. The first argument is `sys.argv[1]`, the next is `sys.argv[2]`, and so on.

Exception Handling

Jython supports exception handling using a try block:

```
try:  
    connect('weblogic','Welcome1',t3://localhost:7001)  
    print '***Connected successfully***'  
except Exception, e:  
    print 'Domain must be running first'  
    print e  
    exit()
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The commands within the try block are executed. If no exceptions are thrown by any commands, then the except block never executes. However, if an exception is thrown, processing within the try block halts and the code in the except block is executed.

Quiz



What is used to determine a block of code in Jython?

- a. A square bracket []
- b. A semicolon
- c. A whitespace
- d. The "begin ... end" words

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: c

Agenda

- WebLogic Scripting Tool (WLST)
- Jython concepts
- WLST concepts
 - WLST modes
 - WLST example
 - Running WLST scripts
 - WLST commands and requirements
- Java Management eXtension (JMX) concepts
- Common WLST tasks
- Fusion Middleware (FMW) commands



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WLST Modes

- Online mode:
 - Connected to a running server
 - Access to all WLS configuration and runtime attributes
 - To create and activate change sessions similar to the WLS console
 - Manage resources, such as starting, stopping, suspending, and migrating
- Offline mode:
 - Domain not running
 - Access to only persisted domain configuration (`config.xml`)
 - To create or update domains similar to using the Configuration Wizard



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

You can use WLST as the command-line equivalent to the Oracle WebLogic Server Administration Console (WLST online) or as the command-line equivalent to the Configuration Wizard (WLST offline).

You can use WLST to connect to a running administration server and manage the configuration of an active domain, view performance data about the resources in the domain, or manage security data (such as adding or removing users). You can also use WLST to connect to managed servers, but you cannot modify configuration data from managed servers. WLST online is a JMX client. It interacts with a server's in-memory collection of MBeans, which are Java objects that provide a management interface for an underlying resource.

Without connecting to a running Oracle WebLogic Server instance, you can use WLST to create domain templates, create a new domain based on existing templates, or extend an existing, inactive domain. However, you cannot use WLST offline to view performance data about the resources in a domain or modify security data (such as adding or removing users). WLST offline provides read and write access to the configuration data that is persisted in the domain's `config` directory or in a domain template JAR that is created using the Domain Template Builder.

WLST Example

```
[oracle@edvmr1p0 /]$ java weblogic.WLST

Initializing WebLogic Scripting Tool (WLST) ...
Welcome to WebLogic Server Administration Scripting Shell
Type help() for help on available commands

wls:/offline> connect('weblogic','password','t3://adminhost:7001')
Connecting to t3://adminhost:7001 with userid system ...
Successfully connected to Admin Server 'myAdmin' that belongs to domain 'mydomain'.

Warning: An insecure protocol was used to connect to the server. To
ensure on-the-wire security, the SSL port or Admin port should be used instead.

wls:/mydomain/serverConfig> cd('Servers')
wls:/mydomain/serverConfig/Servers> ls()
dr-- myAdmin
dr-- myserver1
dr-- myserver2

wls:/mydomain/serverConfig/Servers> cd('myserver1')
wls:/mydomain/serverConfig/Servers/myserver1> get('StartupMode')
'RUNNING'
wls:/mydomain/serverConfig/Servers/myserver1> exit()

Exiting WebLogic Scripting Tool.

[oracle@edvmr1p0 /]$
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

In this example, the administrator:

1. Starts WLST (case-sensitive)
2. Connects to the administration server “myAdmin”
3. Changes to the Servers MBean directory
4. Checks the directory to see which servers exist
5. Changes to the directory of the myserver1 MBean
6. Checks the value of the `StartupMode` attribute for the server “myserver1”
7. Exits WLST

Running WLST Scripts

- Use the `setWLSEnv` script to initialize the `PATH` and `CLASSPATH` required for `WLST`.
 - If no script file is supplied, `WLST` runs in interactive mode.
- Use the `execfile()` command to run additional scripts.

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

To support SSL connection to a server:

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

Command Line



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

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

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

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

WLST Development Tools

There are some useful tools to help write WLST scripts:

Tool	Description
configToScript() (WLST)	WLST provides this command which parses an existing domain and generates the WLST script to recreate the domain.
Script recording (WLST)	WLST provides commands to capture interactive commands to a file: <code>startRecording(recordFilePath, [recordAll])</code> <code>stopRecording()</code>
Script recording (administration console)	Records administration console configuration actions and writes the equivalent WLST commands to a file
Oracle Enterprise Pack for Eclipse (OEPE)	Provides an integrated environment for developing, running, and debugging WLST scripts



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WLST is challenging to new users. This is especially true of administrators who may not be used to coding. Sometimes, it is difficult to tell what is going wrong when you run your scripts.

configToScript()

- The configToScript WLST command:
 - Takes an existing domain as input
 - Generates a WLST script that connects to the domain and re-creates the entire configuration
 - Creates a corresponding property file for commonly changed parameters, such as the domain name, server name, port, and administrative password
 - Creates a separate encrypted file to store any encrypted password attributes
- Use WLST to generate a domain bootstrap script:

```
configToScript('/u01/domains/mydomain',
               'init_mydomain.py')
```

WLST Command Line



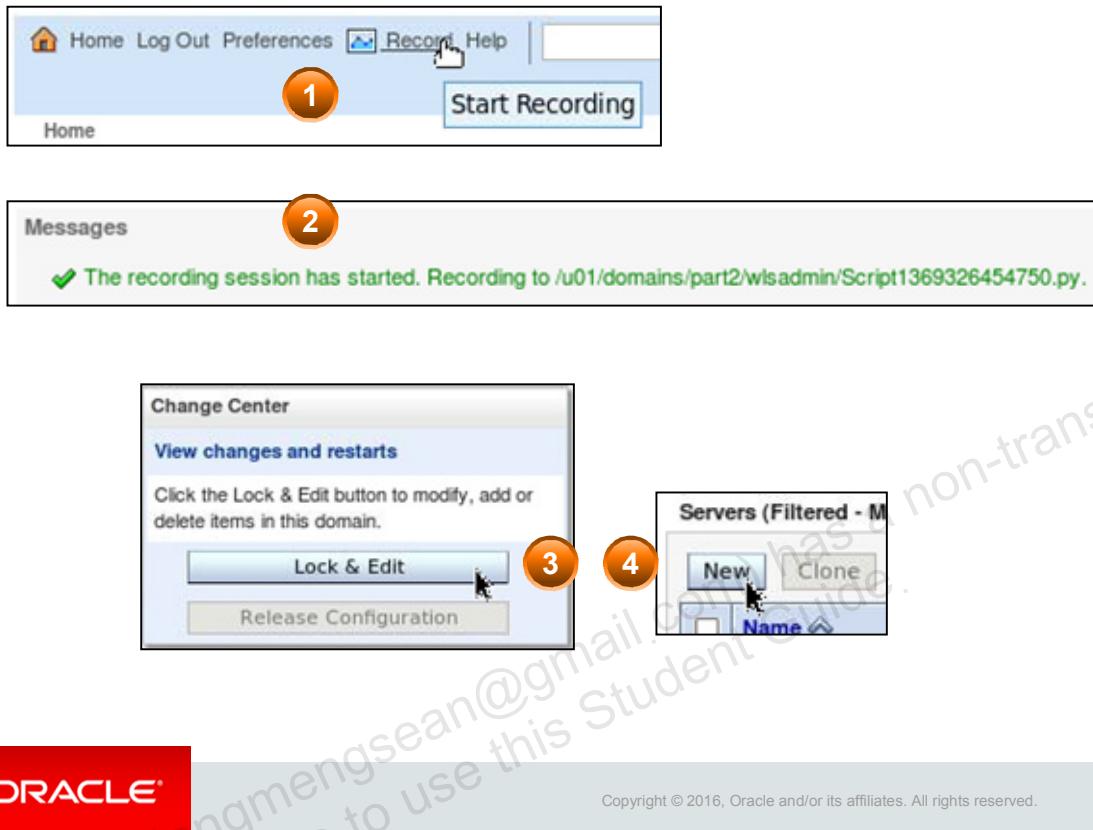
Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This WLST command converts an existing domain configuration (/config directory) into an executable WLST script. You can then use the resulting script to help re-create the domain resources on other machines, or restore a domain to a previously known state.

The command can be run while the domain is running or offline. It generates the following files:

- A WLST script that contains the commands needed to re-create the configuration
- A properties file that contains domain-specific values, including the location of the administration server and the required credentials to access it. You can update the values in this file to update another domain with the same configuration.
- A user configuration file and an associated key file to store encrypted attributes, such as passwords. The user configuration file contains the encrypted information. The key file contains a secret key that is used to encrypt and decrypt the encrypted information.

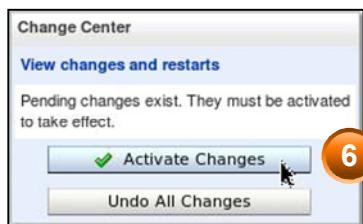
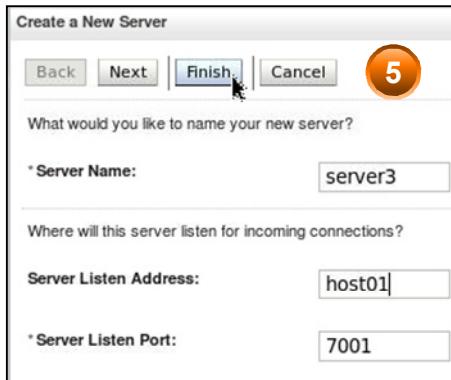
Script Recording Using the Administration Console



Perform the following steps to record administration console activities in a WLST script:

1. Click the Record button to turn on recording.
2. You should see a message that recording is started and where the file containing the script is located.
3. Click Lock & Edit to start an edit session.
Note: You can't click the Record button unless an edit session has already been started. However, if recording was started previously, then it stays active between edit sessions.
4. Configure something using the console. In this example, we create a new server.

Script Recording Using the Administration Console



```
startEdit()  
  
cd('/')  
cmo.createServer('server3')  
  
cd('/Servers/server3')  
cmo.setListenAddress('host01')  
cmo.setListenPort(7001)  
  
activate()
```

Script1369326454750.py

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

5. The server's properties are configured, server name, listen address, and listen port. Click Finish to create the server.
6. Click Activate Changes to realize the configuration change.
7. View the script recording file to find the WLST script used to perform the task.

Oracle Enterprise Pack for Eclipse

```
14
15 #Conditionally import wlstModule only when script is executed
16 if __name__ == '__main__':
17     from wlstModule import * #@UnusedWildImport
18
19 #----- Connect to server -----
20@def connectWLS(host):
21     #WLS Must be running for this script to work
22     try:
23         connect('weblogic', 'Welcome1', host)
24         print "***Connected successfully***"
25     except:
26         print 'Domain ' + host + ' must be running first'
27         exit()
28
29 #----- Create Roles -----
30@def createPolicy():
31     PyDev breakpoint()
32
33     #cmo.getSecurityConfiguration().getRealm().lookup()
```

Manage importing
WLST modules when
running in Jython.

Set breakpoints to stop
script execution so you
can see inside the script.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The Oracle Enterprise Pack for Eclipse provides a WLST scripting facet that leverages the PyDev extension that provides support for developing, executing, and debugging WLST scripts. Using the IDE, you gain the benefits of:

- Syntax and color highlighting and automatic syntax checking
- Auto-completion of methods
- An execution environment
- Real-time debugging with breakpoints and all the features you would expect from a debugger
- An MBean explorer that lets you see the paths to the MBeans that you need to configure
- A built-in interactive mode that includes command history and auto-completion, as well as automatic command line set up with only entering a partial command
- WLST integrated help
- WLST templates

WLST Command Tips

- Use case-sensitive names and arguments of commands.
- Use string literal arguments enclosed within single or double quotation marks.
- Precede the quoted string with `r` while specifying a backslash (\) in the string.
 - Example: `readTemplate (r'c:\mytemplate.jar')`
- Do not use the following invalid characters in object names:
 - Period (.)
 - Slash (/)
 - Backslash (\)
- Use the display help:
 - Example:
`wls:/mydomain/serverConfig> help('disconnect')`



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Note: If you need to change directory (`cd`) to an object name that includes a slash (/) in its name, include the configuration object name in parentheses. For example:
`cd ('JMSQueue/ (jms/REGISTRATION_MDB_QUEUE) ')`

You cannot access security information through WLST while updating a domain. When you use WLST and a domain template, you can create and access security information only when you create a new domain.

General WLST Commands

Many of these commands take additional parameters.

Command	Description
<code>help()</code>	Get help for a given WLST command.
<code>exit()</code>	Quit WLST.
<code>dumpVariables()</code>	Display all variables used by WLST.
<code>dumpStack()</code>	Display the stack trace for the last error that occurred in WLST.
<code>redirect() / stopRedirect()</code>	Redirect all WLST output to a file.
<code>cd()</code>	Navigate from one MBean in the hierarchy to another MBean.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

See the WLS documentation for a complete list of available WLST commands, with examples for each.

To display information about the WLST commands and variables, enter the `help` command. If you specify the `help` command without arguments, WLST summarizes the command categories. To display information about a particular command, variable, or command category, specify its name as an argument to the `help` command. To list a summary of all online or offline commands from the command line, use the following commands, respectively:

```
help('online')
help('offline')
```

The `help` command supports a query. For example, `help('get*')` displays the syntax and usage information for all commands that begin with `get`.

To redirect WLST information, error, and debug messages from standard out to a file, enter:

```
redirect(outputFile, [toStdOut])
stopRedirect()
```

This command also redirects the output of the `dumpStack()` and `dumpVariables()` commands.

Offline WLST Commands

Command	Description
createDomain()	Create a domain by using a given template.
readDomain()	Open an existing domain on the file system.
readTemplate()	Open an existing domain template.
addTemplate()	Apply a template file to the current domain.
updateDomain()	Save changes to the current domain.
writeDomain()	Save changes to the current domain to a specified directory.
writeTemplate()	Save the current domain to a template file.
assign() / unassign()	Target applications or services to servers.
setOption()	Configure domain creation options (domain name, Java home, start mode, and so on).



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WLST offline provides read and write access to the configuration data that is persisted in the domain's config directory or in a domain template JAR that is created using the Domain Template Builder. This data is a collection of XML documents and expresses a hierarchy of management objects. The offline commands include:

- createDomain(domainTemplate, domainDir, user, password)
- readDomain(domainDirName)
- readTemplate(templateFileName)
- addTemplate(templateFileName)
- writeTemplate(templateName)
- assign(sourceType, sourceName, destinationType, destinationName)
- setOption(optionName, value)

WLST also includes the `configToScript` command that reads an existing domain and outputs a WLST script that can re-create the domain. Set the password for the default user, if it is not already set. The default username and password must be set before you can write the domain template. For example:

```
cd('/Security/domainname/User/username')
cmo.setPassword('password')
```

Online WLST Commands

Command	Description
connect()	Connect to a server by using supplied credentials.
disconnect()	Disconnect from the current server.
shutdown()	Shut down servers.
start()	Use the Node Manager to start servers.
startEdit()	Begin a new change session.
stopEdit()	Release the edit lock and discard any changes.
activate()	Commit all changes in the current session.
showChanges()	List all changes made in the current session.
isRestartRequired()	Determine if any changes require a server restart.
deploy() / redeploy()	Deploy an application to servers.
undeploy()	Shut down a running application on servers.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Learning WLST initially is sometimes overwhelming for somebody that is new to WLST and Jython. The administration console provides a script recording tool that works when the server is running. This allows an administrator to perform operations within the console and have the corresponding WLST commands written to a text file in the domain's root folder. This makes it easy to create new WLST scripts. The open source project, wlnav, is another way to work with WebLogic MBeans and learn WLST.

Quiz



Using WLST's _____ mode, you can supply commands one at a time and get immediate feedback.

- a. Management
- b. Operational
- c. Sequential
- d. Template
- e. Interactive

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: e

In Interactive mode, you can quickly prototype or troubleshoot some WLST commands.

Agenda

- WebLogic Scripting Tool (WLST)
- Jython concepts
- WLST concepts
- Java Management eXtension (JMX) concepts
 - JMX overview
 - Configuration and runtime mbeans
 - WebLogic Server MBean examples
 - Browsing MBean documentation
 - Referencing MBeans in WLST
- Common WLST tasks
- Fusion Middleware (FMW) commands

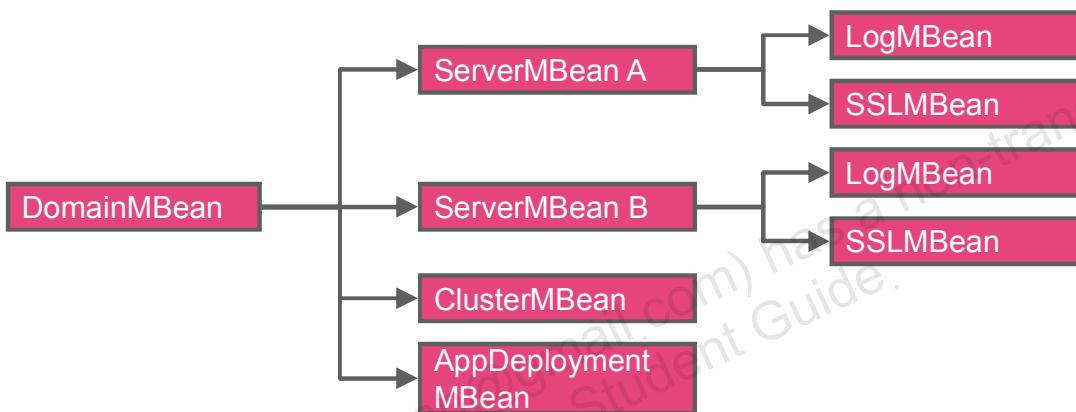


Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic JMX Overview

JMX MBeans:

- Are hierarchical Java objects found on the server
- Have attributes and operations
- Support the configuration, management, and monitoring of all types of server resources



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

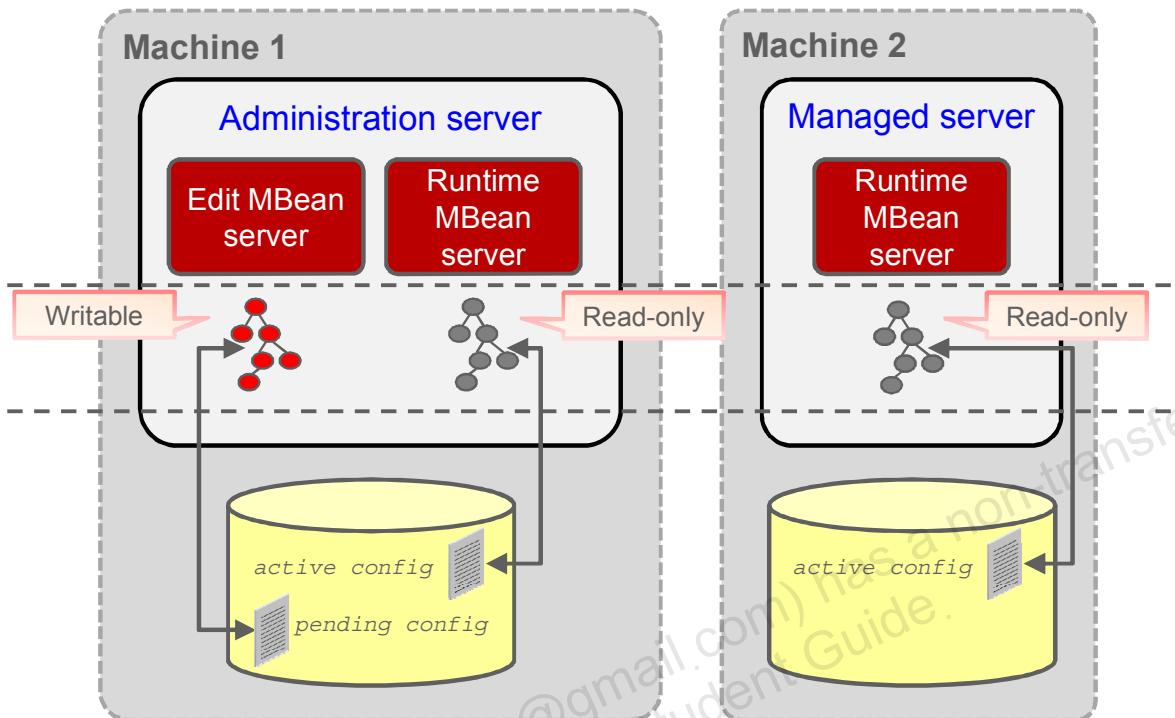
This diagram shows the hierarchical structure of MBeans.

The JMX specification does not impose a model for organizing MBeans. However, because the configuration of an Oracle WebLogic Server domain is specified in an XML document, Oracle WebLogic Server organizes its MBeans into a hierarchical model that reflects the XML document structure. For example, the root of a domain's configuration document is `<domain>` and below the root are child elements such as `<server>` and `<cluster>`. Each domain maintains a single MBean of the DomainMBean type to represent the `<domain>` root element. Within DomainMBean, the JMX attributes provide access to the MBeans that represent child elements such as `<server>` and `<cluster>`.

All Oracle WebLogic Server MBeans can be organized into one of the following general types:

- Runtime MBeans contain information about the runtime state of a server and its resources. They generally contain data only about the current state of a server or resource, and they do not persist this data. When you shut down a server instance, all runtime statistics and metrics from the runtime MBeans are lost.
- Configuration MBeans contain information about the configuration of servers and resources. They represent the information that is stored in the domain's XML configuration documents.

Configuration MBeans



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

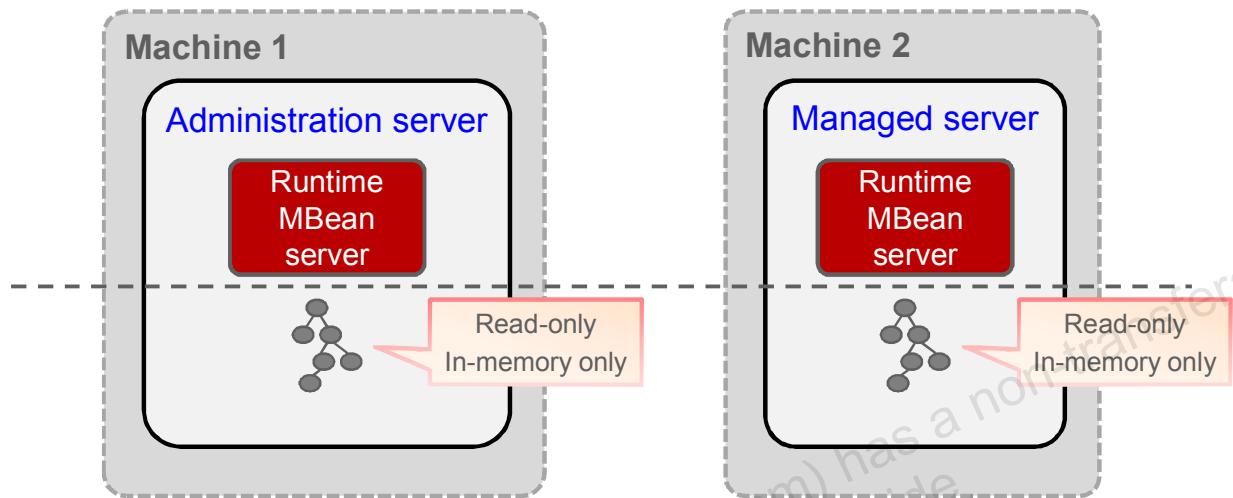
This graphic depicts MBeans reading and writing to the config.xml files. Each domain describes its configuration in an XML document that is located in the domain's configuration directory. At run time, each Oracle WebLogic Server instance in a given domain creates an in-memory representation of the configuration described in this document. The in-memory representation of a domain's configuration is a collection of read-only managed beans (MBeans) called Configuration MBeans.

In addition to the read-only Configuration MBeans, the administration server maintains another collection of Configuration MBeans that you can edit. To edit these Configuration MBeans, you use a JMX client (either the Administration Console, WLST, or a client that you create) to obtain a lock.

While you have the lock on the editable Configuration MBeans, you can save your in-memory changes, which causes the administration server to write the changes to a set of pending configuration documents in the domain directory. The Oracle WebLogic Server instances do not consume (commit) the changes until you activate them.

When you activate the changes, each server in the domain determines whether it can accept the change. If all the servers can accept the change, they update their copy of the domain's configuration document. Then they update their working copy of the Configuration MBeans and the change is completed.

Runtime MBeans



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

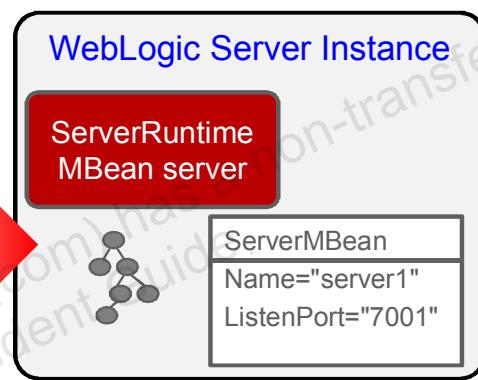
Just as Configuration MBeans represent the configuration of a WebLogic domain, Runtime MBeans represent the running state of all the running components in a WebLogic domain. Unlike Configuration MBeans, Runtime MBeans are not persisted to disk and are only kept in memory for monitoring purposes. If a server shuts down, then any Runtime MBean data that resided on the server is no longer available. Runtime MBeans are used to capture and monitor the historic and current state of the running server. This information is used to tune and troubleshoot your running application.

WebLogic Server MBean Examples

	Properties	Description
Name	ServerMBean	Represents the configuration of a WebLogic Server
Attributes	ListenAddress	Represents the listening host address of this server
	ListenPort	Represents the listening port of this server
Operations	isSet (property)	Returns true if the specified attribute is set

```
<domain>
  <server>
    <name>server1</name>
    <listen-port>7011</listen-port>
  </server>
</domain>
```

config.xml



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

MBean Properties in the Administration Console

The screenshot shows the 'Settings for server1' page in the WebLogic Administration Console. The 'Configuration' tab is selected. Under the 'General' section, there is a 'Listen Port' field set to '7011'. A red box highlights this field, and a callout with the number '1' points to a tooltip that says: 'The default TCP port that this server uses to listen for regular (non-SSL) incoming connections. More Info...'. Another callout with the number '2' points to the help text for the 'Listen Port' field, which reads: 'The default TCP port that this server uses to listen for regular (non-SSL) incoming connections. Administrators must have the right privileges before binding to a port or else this operation will not be successful and it will render the console un-reachable. If this port is disabled, the SSL port must be enabled. Additional ports can be configured using network channels. The cluster (multicast) port is configured separately.'

This slide shows the same settings in the WebLogic administration console and how to find the corresponding MBean attribute using the administration console help:

1. Next to the property field in the administration console, there is a short description of the field. There is a *More Info...* link that opens the administration console help to a more detailed description of that field.
2. This is the administration console help page for Listen Port. It shows the associated MBean attribute name, `ServerMBean.ListenPort`, which is used to reference this field from within WLST.

Browsing MBean Documentation

The screenshot shows the Oracle Fusion Middleware Documentation interface for Oracle WebLogic Server. At the top, the Oracle logo is visible, followed by the title "Oracle Fusion Middleware Documentation" and a dropdown menu set to "12c (12.1.2)". Below this, a breadcrumb navigation shows "Home > 12c Release 1 (12.1.2)". The main content area is titled "Oracle WebLogic Server". A horizontal navigation bar below the title includes "View by:" followed by "Get Started", "Develop and Deploy", "Administer", "Secure", "Monitor and Tune", and "Reference". An arrow points from the "Reference" button to a callout box. The callout box contains the heading "Programming APIs and MBeans" and a bulleted list of links:

- Java Platform EE 6 API specification
- Java SE 6 API specification
- Java API Reference for WebLogic Server
- JMS C API Reference for WebLogic Server
- WebLogic Messaging .NET API Reference for WebLogic Server
- MBean Reference for WebLogic Server
- WebLogic Web Services Reference

At the bottom of the page, the Oracle logo is on the left, and the copyright notice "Copyright © 2016, Oracle and/or its affiliates. All rights reserved." is on the right.

Browsing MBean Documentation

The screenshot shows the 'WEBLOGIC SERVER MBEAN REFERENCE' interface. On the left, a tree view lists various MBeans under 'The WebLogic Server MBean Reference'. A red box highlights the 'BasicRealmMBean' node under 'Domain Configuration MBeans'. A mouse cursor is positioned over this node. A callout bubble points to the right panel with the text: 'Select an MBean from the left panel and see its details in this panel.' In the center-right panel, the title 'BasicRealmMBean' is displayed above the 'Overview' section. The 'Overview' section contains a detailed description of the MBean. Below it are sections for 'Attributes' and 'Operations'. A red box highlights the 'Attributes' section. Another callout bubble points to this section with the text: 'Organized list of all WebLogic MBeans'. At the bottom of the right panel, there is a table with three rows: 'Fully Qualified Interface Name', 'Factory Methods', and 'Subtypes'. The 'Fully Qualified Interface Name' row contains the value 'weblogic.management.configuration.BasicRealmMBean'. The 'Factory Methods' row states 'No factory methods. Instances of this MBean are created automatically.'. The 'Subtypes' row lists 'The following MBeans extend or implement this MBean type:' followed by two items: 'CachingDisabled' and 'ObjectName'. The bottom of the page features an 'ORACLE' logo and a copyright notice: 'Copyright © 2016, Oracle and/or its affiliates. All rights reserved.'

Referencing MBeans in WLST

Use the `getMBean()` API to assign an MBean to a variable and reuse that variable to manage the MBean.

```
#Create channel on server1
server=getMBean('/Servers/server1')
server.createNetworkAccessPoint('RepChannel')
channel=
getMBean('/Servers/server1/NetworkAccessPoints/RepChannel')
channel.setProtocol('t3')
channel.setListenAddress('host01')
channel.setListenPort(5000)
channel.setEnabled(true)
```

CreateReplication.py



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Quiz



WLST communicates with Oracle WebLogic Server's _____ to retrieve and update resources on a running server.

- a. Templates
- b. Logs
- c. MBeans
- d. Scripts

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: c

WLST is based on JMX, which supports remote server management through MBean objects.

Agenda

- WebLogic Scripting Tool (WLST)
- Jython concepts
- WLST concepts
- Java Management eXtension (JMX) concepts
- Common WLST tasks
 - Creating and modifying templates and domains
 - Connecting to, configuring, and monitoring a server
 - Adding a server to a cluster
 - Creating and monitoring a data source
 - Creating an LDAP authentication provider
 - Deploying an application
- Fusion Middleware (FMW) commands



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Creating a Template and a Domain

```
readTemplate('mybasetemplate.jar')          Reads in a template jar file  
  
setOption('DomainName','mydomain')  
setOption('JavaHome','/home/myjdk')  
setOption('ServerStartMode','prod')          Sets some options  
  
writeDomain('/home/mydomains')  
closeTemplate()                            Writes a domain based on the template  
                                         and the changes and closes the template  
  
readDomain('/home/mydomains/mydomain')  
addTemplate('myjms.jar')  
addTemplate('myapps.jar')  
updateDomain()  
closeDomain()  
exit()
```

Reads in the domain, extends it with some templates, and updates the domain



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

See the WLS documentation for the `setOption` command to view a list of available domain creation options.

The following example writes a domain configuration to a domain template:

```
readDomain('/home/mydomains/mydomain')  
writeTemplate('myTemplate.jar')
```

Connecting to a Server

Administrator username

Administrator password

URL of server

```
connect ('weblogic','Welcome1',t3://adminhost:7001)
```

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The connect WLST command is used to connect to a running WebLogic Server.

Password Management

To avoid using plain text credentials in your scripts, do one of the following:

- Require them as command-line arguments.
- Prompt the user to enter them.
- Create and use an encrypted password file.

Create encrypted password file for previously used credentials:

```
connect(username, password, myurl)  
storeUserConfig()
```

Stored in OS user's home directory by default

Connect to a server using the current password file:

```
connect(url=myurl)  
...
```

File retrieved from the home directory by default



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The `storeUserConfig()` command creates a user configuration file and an associated key file for the identity of the current user you are connected to WLS with. The user configuration file contains an encrypted username and password. The key file contains a secret key that is used to encrypt and decrypt the username and password. Only the key file that originally encrypted the username and password can be used to decrypt the values. If you lose the key file, you must create a new user configuration and key file pair. If you do not specify file names or locations, the command stores the files in your home directory as determined by your Java Virtual Machine (JVM). The location of the home directory depends on the SDK and type of operating system on which WLST is running. The default file names are `<username>-WebLogicConfig.properties` and `<username>-WebLogicKey.properties`.

If you do not specify credentials as part of the `connect` command, and a user configuration and a default key file exist in your home directory, then the command uses the credentials found in those files. This option is recommended if you use WLST in script mode, because it prevents you from storing unencrypted user credentials in your scripts. Alternatively, you can provide the specific locations and names of these files by using the `userConfigFile` and `userKeyFile` command arguments.

WLST Variables

Some common WLST variables:

Variable	Description
cmo	Current Management Object. This variable is automatically set to the configuration MBean instance to which you navigate. <code>cmo.create('myServer', 'Server')</code>
domainName	The name of the domain to which WLST is connected.
domainRuntimeService	Represents the DomainRuntimeServiceMBean, which is only available when connected to the administration server. <code>domainRuntimeService.getServerRuntimes()</code>
runtimeService	Represents the RuntimeServiceMBean. <code>runtimeService.getServerRuntime()</code>
editService	Represents the EditServiceMBean, which is only available when connected to the administration server. <code>editService.getDomainConfiguration()</code>
exitonerror	A Boolean value that indicates whether WLST terminates script execution when it encounters an exception.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Password Management

Password change using WLST:

```
connect('weblogic','Welcome1','t3://host01.example.com:7001')  
atnr=cmo.getSecurityConfiguration().getDefaultRealm().  
    lookupAuthenticationProvider("DefaultAuthenticator")  
  
atnr.changeUserPassword('weblogic','Welcome1','Welcome2')  
disconnect()  
  
connect('weblogic','Welcome2','t3://host01.example.com:7001')  
exit()
```

The `cmo` built-in WLST variable is used to obtain a reference to the Default Authentication Provider.

The `changeUserPassword` method is invoked to change the password for the `weblogic` user.

A connection is attempted, using the password that was just reset.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

CMO is the WLST built-in variable. CMO stands for Current Management Object. While programming in WLST you can use `cmo` to point to the current MBean (object) instance you are navigating into.

The example shows how to obtain the Authentication Provider Object by referencing CMO. With a handle to the Default Authentication Provider it is possible to call the `changeUserPassword()` function, which requires username, current password and new password as parameters, to change the password for the `weblogic` user.

Configuring a Server

```
newServerName = 'server1'           Variable declarations
newServerPort = 7011
machineName = 'machine1'

edit()
startEdit()                         Start a configuration edit session.

server = cmo.create(newServerName, 'Server')
server.setListenPort(newServerPort)
server.setMachine(getMBean('/Machines/' + machineName))

save()                                Save and activate all changes.
activate(block='true')
print 'Server created successfully.'
exit()
```

Create a new server and then set its port and machine.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script shows the basics on how to create and configure a WebLogic server.

Monitoring a Server

```
serverRuntime()
threadPool = getMBean('/ThreadPoolRuntime/ThreadPoolRuntime')
webModule = getMBean('/ApplicationRuntimes/' + appName +
                      '/ComponentRuntimes/' + serverName +
                      '/' + appWebRoot)

while 1: Loop indefinitely.
    throughput = threadPool.getThroughput()
    appStatus = webModule.getStatus() Gather runtime information for the server's current statistics.
    appCurrSessions = webModule.getOpenSessionsCurrentCount()

    print '%.1f\t%'%(throughput) + appStatus + '\t' +
          str(appCurrSessions) Print statistics.

    java.lang.Thread.sleep(1000)

Sleep for one second and then loop again.
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script shows an example of monitoring some server statistics and displaying them to standard out.

Adding a Server to a Cluster

```
edit()
startEdit()

cluster = cmo.create('cluster1', 'Cluster')
cluster.setClusterMessagingMode('unicast')           Create a new cluster and
                                                    set its broadcast protocol.

server = getMBean('/Servers/server1')
server.setCluster(cluster)
server = getMBean('/Servers/server2')
server.setCluster(cluster)                         Add two configured
                                                    servers to the cluster.

save()                                              Save and activate all changes.

activate(block='true')
print 'Cluster created successfully.'
exit()
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script shows how to create a cluster and add two existing servers to it.

Creating a Data Source

```
edit()  
startEdit()           Start a configuration edit session.  
  
targetServer = getMBean('/Servers/server1')          Get MBean object  
                                                       for server1.  
  
jdbcSysRes = cmo.create('MyXADataSource', 'JDBCSystemResource')  
jdbcResource = jdbcSysRes.getJDBCResource()  
jdbcResource.setName('MyXADataSource')               Create data source  
                                                       and set its name.  
  
jdbcResParams = jdbcResource.getJDBCDataSourceParams()  
jdbcResParams.setJNDINames(['jdbc/MyXADataSource'])  
jdbcResParams.setGlobalTransactionsProtocol('TwoPhaseCommit')  
  
connectionPool = jdbcResource.getJDBCConnectionPoolParams()  
connectionPool.setInitialCapacity(5)                 Set connection pool  
connectionPool.setMaxCapacity(10)                     properties.  
connectionPool.setMinCapacity(1)
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script shows how to create a data source, set some of the most important properties, and configure its associated connection pool.

Creating a Data Source

```
dr = jdbcResource.getJDBCDriverParams()
dr.setDriverName('oracle.jdbc.xa.client.OracleXADataSource')
dr.setUrl('jdbc:oracle:thin:@localhost:1521:orcl')
dr.setPassword('Welcome1')
driverProperties = dr.getProperties()
userProperty = driverProperties.createProperty('user')
userProperty.setValue('wlsdata')

jdbcSystemResource.addTarget(targetServer)

save()
activate(block='true')
print 'Data Source created successfully.'
exit()
```

Configure driver properties.

Target the data source to a server.

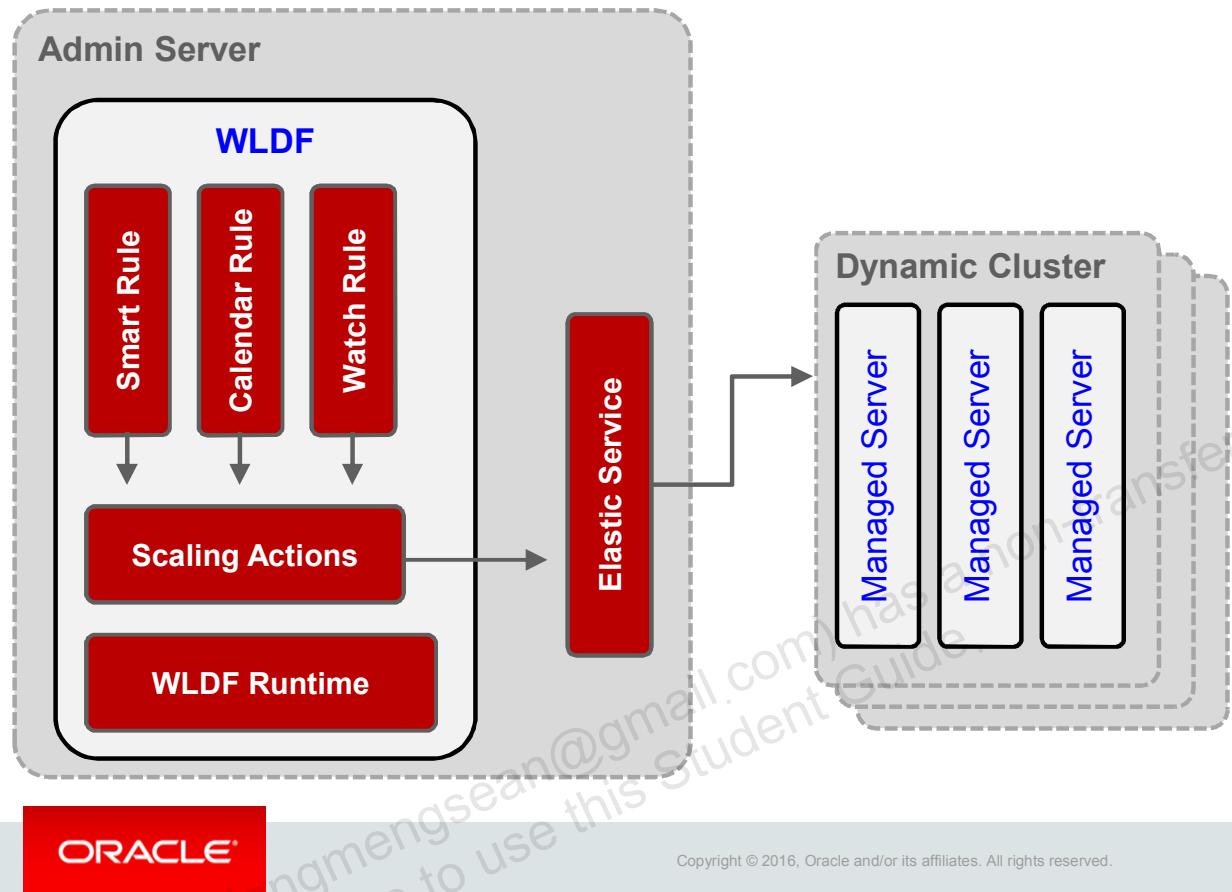
Save and activate all changes.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script continues to show how to configure the JDBC driver for the data source, and target it for a server.

Dynamic Cluster Scaling



Dynamic Cluster Scaling

Scaling of a dynamic cluster can be achieved by following means:

- On-demand through WLS Administration Console and WLST
- Using an automated calendar-based schedule utilizing WLDF policies and actions
- Using an automated WLDF policies based on performance metrics

Monitoring a Data Source

```
serverRuntime()          Navigate to the server runtime MBean tree.  
ds = getMBean('/JDDataSourceRuntime/server1' +  
             '/JDDataSourceRuntimeMBeans/MyXADatasource')  
  
print 'CAP' + '\t' + 'TOTAL' + '\t' + 'ACTIVE' + '\t' +  
      'AVAIL' + '\t' + 'LEAK'  
  
while 1:                Loop indefinitely.  
    cap = ds.getCurCapacity()  
    total = ds.getConnectionsTotalCount()  
    active = ds.getActiveConnectionsCurrentCount()  
    avail = ds.getNumAvailable()  
    leak = ds.getLeakedConnectionCount()  
    print str(cap) + '\t' + str(total) + '\t' + str(active) +  
          '\t' + str(avail) + '\t' + str(leak)  
    java.lang.Thread.sleep(2000)  Sleep for two seconds and then loop again.
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script shows one way to monitor a data source within a loop.

Creating an LDAP Authentication Provider

Get the default security realm MBean.

```
realm = cmo.getSecurityConfiguration().getDefaultRealm()

provider = realm.createAuthenticationProvider('MyLDAPProvider',
'weblogic.security.providers.authentication.LDAPAuthenticator')

Create an LDAP authentication provider named MyLDAPProvider.

provider.setControlFlag('SUFFICIENT')
provider.setHost('localhost')
provider.setPort('389')
provider.setPrincipal('cn=Directory Manager')
provider.setCredential('Welcome1')
provider.setUserBaseDN('dc=example,dc=com')
provider.setUserNameAttribute('uid')
provider.setGroupBaseDN('dc=example,dc=com')
provider.setStaticGroupNameAttribute('cn')
```

Sets some basic settings required for the provider



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script shows how to create and configure an LDAP authentication provider. Note that the change management commands are not included in this script, but are still required for starting an edit session and activating your changes.

Modifying a Domain Offline

```
readDomain('/u01/app/domains/wlsadmin')  
  
currentMachine = cmo.create('machine1','UnixMachine')  
cd('/Machines/machine1')  
nodeManager = cmo.create('machine1','NodeManager')  
nodeManager.setListenAddress('myHostName')  
nodeManager.setListenPort(5556)  
  
for currentIndex in range(1, 5):  
    currentServerName = 'server' + str(currentIndex)  
    cd('/Servers/' + currentServerName)  
    cmo.setMachine(currentMachine)  
  
updateDomain()  
print ''  
print 'Domain updated successfully.'
```

Read in the original domain.

Create a machine and set some properties.

Loop through server1 through server5 and set each on the new machine.

Write out the domain with a new configuration.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script shows you how to read in an existing domain in offline mode, modify it, and update the original domain by writing it back to disk.

Deploying an Application

```
deploy(appName='MyApp',  
       path='/apps/MyApp.ear',  
       targets='server1')
```

Deploy application named MyApp

Found in this location

To this server



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script shows an example of deploying an Enterprise Application to a WebLogic Server instance.

Creating a Dynamic Cluster

```
cd('/')
cmo.createServerTemplate('mytemplate')          Create a Server Template.

cd('/ServerTemplates/mytemplate')
cmo.setListenPort(7100)                          Set Listening Port.

cd('/ServerTemplates/mytemplate/SSL/mytemplate')
cmo.setListenPort(8100)                          Set SSL Listening Port.

cd('/ServerTemplates/'mytemplate')
cmo.setMachine(None)                           Assign a machine to the
                                                Server Template.

cd('/')
cmo.createCluster('mycluster')                  Create a new cluster.

cd('/Clusters/mycluster')
cmo.setClusterMessagingMode('unicast')          Set messaging mode.
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script shows you how to create a Dynamic Cluster.

Creating a Dynamic Cluster

```
cd('/ServerTemplates/mytemplate')
cmo.setCluster(getMBean('/Clusters/'mycluster'))  
  
cd('/Clusters/mycluster/DynamicServers/mycluster')
cmo.setServerTemplate(getMBean('/ServerTemplates/mytemplate'))  
  
cmo.setDynamicClusterSize(1)
cmo.setMaxDynamicClusterSize(6)  
  
cmo.setCalculatedListenPorts(true)
cmo.setCalculatedMachineNames(true)  
  
cmo.setDynamicClusterCooloffPeriodSeconds(120)
cmo.setCalculatedListenPorts(true)
```

Bind the Cluster to the Server Template.

Set the initial Cluster size.
Set the max Cluster size.

Enable dynamic port and machine name assignation to servers.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script shows you how to create a Dynamic Cluster.

Scaling Up a Dynamic Cluster

```

connect('username', 'password', 'connect url') Connect to Admin Server.
start('mycluster', 'Cluster') Start the cluster
state('mycluster') And verify its status
There are 1 server(s) in cluster: mycluster

States of the servers are
mycluster-server-1---RUNNING

scaleUp('mycluster', 3, true, true) Call the scaleUp command.
Remote ScaleUp started successfully after 24 seconds
All servers are now running.
.
.
state('mycluster')
There are 4 server(s) in cluster: mycluster Verify the status of the cluster.
States of the servers are
mycluster-server-1---RUNNING
.
.
mycluster-server-4---RUNNING

```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This examples shows you how to scale up a Dynamic Cluster.

The example connects to WebLogic Server, starts the cluster and then uses the `state` command to show the current cluster state.

The `scaleUp` command is then used to scale up the cluster.

`Scale up` takes up to 6 parameters, the first 2 required.

- `clusterName` – Required the cluster to scale up
- `numServers` – Required, the number of servers to be started
- `updateConfiguration` – Optional, value specifying whether WLST should increase the maximum size of the cluster if there are not enough non-running servers. Defaults to false.
- `block` – Optional. Should WLST block until the operation completes and the new servers are started. Defaults to true.
- `timeoutSecs` – Optional time out, in seconds, for how long WLST to should wait before cancelling the operation. Default 600s.
- `type` – Optional is specified must be `Dynamic Cluster`

Scaling Down a Dynamic Cluster

```
scaleDown('mycluster', 3, true, true) Scale down the cluster by 3.  
Remote ScaleDown started successfully after 3 seconds.  
The servers were stopped successfully.  
  
state ('mycluster')  
There are 1 server(s) in cluster: mycluster  
  
States of the servers are Verify the status of the  
mycluster-server-1---RUNNING cluster.
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

This script shows you how to scale down a Dynamic Cluster. Scale down is the inverse of scaleUp and takes the same parameters. In this example a dynamic cluster of 4 is scaled down by three to a size of 1.

Quiz



The WSLT built-in variable which identifies the current JMX MBean is called:

- a. cto(Current Transaction Object)
- b. cdo (Current Data Object)
- c. cmo (Current Management Object)
- d. cwo (Current WebLogic Object)

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: c

Agenda

- WebLogic Scripting Tool (WLST)
- Jython concepts
- WLST concepts
- Java Management eXtension (JMX) concepts
- Common WLST tasks
- Fusion Middleware (FMW) commands



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Some FMW Commands

FMW Product	Command	Description
SOA Suite	<code>sca_deployComposite()</code>	Deploy a SOA composite application.
WebCenter	<code>listJCRPortalConnections()</code>	List all Oracle Portal connections configured for WebCenter applications.
ADF	<code>adf_createHttpURLConnection()</code>	Create a new ADF URL connection.
JRF	<code>applyJRF()</code>	Configure a managed server or cluster with Java Required Files applications and services.
OAM	<code>createOAMIdentityAsserter()</code>	Create a new identity asserter.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Some FMW products have custom WLST scripts that are installed when the respective product is installed. This slide shows some examples of these commands.

Note: There are far too many custom FMW scripts to mention in this course. A reference of the commands is found in the *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference 11g Release 1*. FMW is not released as version 12c yet so all commands are currently listed in the 11g guide as of the time of this writing. Remember that these commands are subject to change when FMW 12c is released. FMW scripts require running within an environment that is specialized for that FMW product area. Each FMW product has its own version of `wlst.sh` that sets the environment properly to work with that product.

Summary

In this lesson, you should have learned how to:

- Run commands in WLST interactive mode
- Write simple WLST scripts
- Run WLST scripts



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Practice 6-1 Overview: Creating and Modifying a Domain with WLST

This practice covers the following topics:

- Creating a new WebLogic domain using WLST
- Starting a domain using WLST
- Connecting to the Administration Server with WLST
- Modifying the domain configuration using WLST



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Practice 6-2 Overview: Monitoring a Domain with WLST

This practice covers the following topics:

- Connecting to a domain with WLST
- Monitoring different subsystems with WLST



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Sean Tay (taysiangmengsean@gmail.com) has a non-transferable
license to use this Student Guide.

RESTful Management Services

RESTful Management Services and WebLogic

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe WebLogic REST capabilities
- Create and execute WebLogic REST requests
- Evaluate WebLogic REST results and responses
- Monitor, manage, and configure WebLogic using REST



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Agenda

- WebLogic Server and REST
- Working with WebLogic REST
- WebLogic REST Operations and Workflows



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

What Is REST?

REpresentational State Transfer (REST):

- Is an architecture style of distributed software
- Is implemented as a client-server solution based on HTTP requests and responses
- Is very loosely coupled and scales well
- Takes advantage of WWW technologies and protocols, including HTTP, XML, and JSON

Using cURL, do an HTTP GET.

```
$ curl -i -X GET  
-H Accept:application/json  
-H X-Requested-By:MyClient  
http://localhost:7001/  
management/weblogic/latest/servers
```

Data is being sent in JSON.

Using a well-defined address and port

For a very specific path



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

REpresentational State Transfer (commonly referred to as *REST*) is a style of software architecture. REST is based on existing standards such as XML, HTTP, JSON, and others to support the transfer of data and object state in well-defined ways. In general, REST is defined by a set of basic concepts that, if adhered to by applications, make the system “RESTful.”

Common REST concepts include:

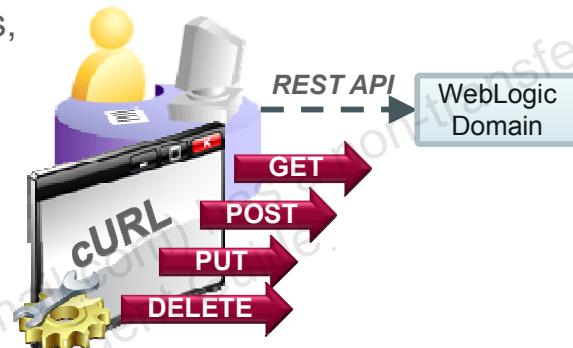
- **Data elements:** Resources, which are usually data objects and resource IDs (addresses and partial URLs), are accessible via standard interfaces such as HTTP.
- **Components:** Servers, clients, gateways, and proxies can all communicate by transferring representations of resources using standard operations such as POST and GET. In fact, REST is often considered more supportive of HTTP than most browser HTTP implementations because it uses other operations such as DELETE, and POST.
- **Stateless:** All requests are stateless. This means that every request must be stand-alone and not dependent on any other request. With REST, a message must include all the information that is required to understand the context of the message.

In the example in the slide, the cURL utility (available natively on UNIX-based operating systems) can be used to make URL requests—in this case, to perform a GET operation to return a set of server information.

WebLogic Server and REST

WebLogic RESTful management services:

- Provide a public interface for configuring, monitoring, deploying to, and administering WebLogic Server
- Are supported in all environments
- Are based on WebLogic Mbeans interfaces
- Support configuration, management, and reporting of:
 - Domains, Partitions, Servers, and other environments
 - Multitenant environments
 - And others



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic RESTful management services provide a comprehensive public interface for configuring, monitoring, deploying, and administering WebLogic Server in all supported environments. This lesson describes the RESTful management services supported by WebLogic Server.

The 12.2.1 release of RESTful management services provides comprehensive support for WebLogic Server administration through the dynamic generation of REST resources based on WLS MBeans and descriptor interfaces. There are resources to support the configuration and monitoring of partitioned and nonpartitioned environments, life cycle management resources, and legacy resources from 12.1.3.

There are two main bean types:

- Configuration: Used to configure WebLogic Server
- Runtime: Used to monitor WebLogic Server and for some operations, control WebLogic Server (for example, starting and stopping servers, and shrinking data source connection pools)

For more information, see **Oracle® Fusion Middleware Administering Oracle WebLogic Server with RESTful Management Services**.

What Can Be Monitored Using WebLogic REST?

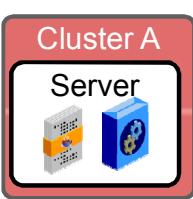
- Clusters: All clusters within a given domain, a specific cluster, or a given member of a cluster
- Servers: All servers within a specific domain, or any given specific server instance
- Applications: All applications deployed in a specific domain or a specific application
- Data sources: All data source running in a domain, or a specific data source
- JMS, Partitions and others

```

{
  "body": {
    "items": [
      {
        "name": "AdminServer",
        "state": "RUNNING",
        "health": "HEALTH_OK"
      },
      {
        "name": "server1",
        "state": "RUNNING",
        "health": "HEALTH_OK"
      },
      {
        "name": "server2",
        "state": "SHUTDOWN"
      }
    ],
    "messages": []
  }
}

```

These resources can also be managed!
Started, stopped, configured, modified
and so on!



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Many areas of a WebLogic Server can be monitored using RESTful management services. Some of the components which can be monitored include:

- Clusters: WebLogic can monitor the overall state of all clusters within a given domain, or a given specific cluster, including all servers within that cluster.
- Server: WebLogic can monitor all servers within a given domain, or any specific server.
- Applications: WebLogic can monitor all applications deployed in a domain or a specific application.
- Data sources: WebLogic can monitor all data sources running in a domain or a specific data source.
- JMS: Monitor JMS Servers, queues, topics and other JMS components
- Partitions: Monitor partitions

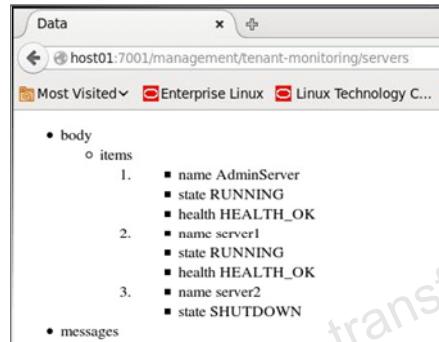
In addition to monitoring resources in the WebLogic Bean tree, covered later in this lesson, they can also be created, started, stopped, modified and otherwise managed using REST. These components represent only a partial list of what can be created, monitored and managed. Using the WLS Bean Tree almost every aspect of WebLogic Server can be monitored and managed.

Clients and WebLogic REST

Clients which can interact with WebLogic REST include:

- cURL and similar tools
- Python
- Java and JavaScript
- Perl
- And many others

*Any client that can
“speak” REST can
interact with
WebLogic REST!*



```

$ ./simple.py
Using URL http://host01:7001/management. . .
. . .
Server: AdminServer [ RUNNING ]
Server: server1 [ RUNNING ]
Server: server2 [ SHUTDOWN ]

```

```

Client c = Client.create();
client.addFilter(new HTTPBasicAuthFilter(...));
WebResource webResource=
c.resource("http://host:7001/management/...");

```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic RESTful management can be accessed via any client which is REST aware and can make URL style calls. Beyond the built-in REST client, any number of possible clients are possible, including:

- Python: Provides support for interacting with URLs via its urllib2 and REST, via its JSON libraries
- Java: Supports URLs via a variety of URL, HTTP connection and JSON classes and libraries. The Sun Jersey libraries provide detailed support for REST.

WebLogic Server also supports a built in client which works with the deprecated tenant-monitoring tree only and can return HTML responses to requests.

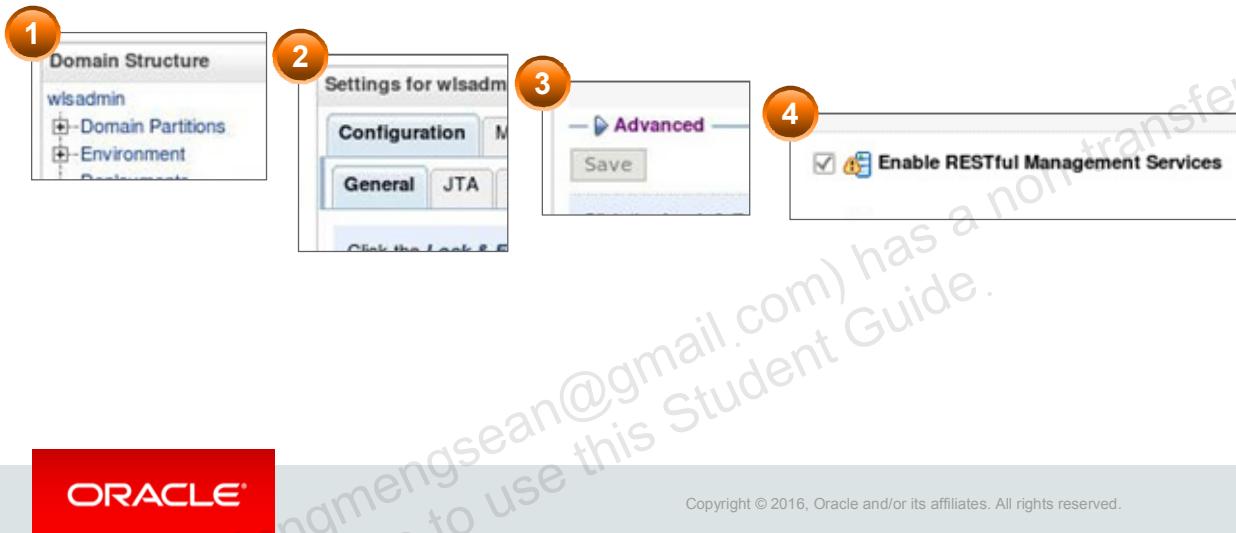
Enabling REST Management

WebLogic Server:

- REST management disabled by default
- Is enabled using the WebLogic Console

To Enable RESTful management:

Changed in 12.2.1 to
enabled by default!



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic Server supports a variety of RESTful management APIs. These APIs can be enabled or disabled. Before WebLogic Server 12.2.1, RESTful management was disabled by default. To enable RESTful management:

0. Log in to the WebLogic Server console and click Lock and Edit (not shown).
1. From the Domain Structure pane select the domain. In this example **wlsadmin**.
2. Select the General tab from within the Configuration Tab.
3. Click Advanced.
4. Select Enable RESTful Management Services.
5. (not shown) Click Save.
6. (Now shown) Click Activate Changes.

Note that you will need to restart the administration server, as indicated by the ! within the triangle symbol next to “Enable RESTful Management Services”.

Multitenant (MT) Versus Nonmultitenant environments

MT environments differ from non-MT environments:

Multitenant Resources	Nonmultitenant Resources
Run in partitions	Run at domain level
Scoped to the partition	Scoped to the domain as a whole
Accessed via a partition-specific URL <code>http://host:port/partition/management</code>	Accessed via a domain-specific URL <code>http://host:port/management</code>
Are limited in scope. Some MBeans are not available.	All MBeans available.
Accessed using partition-level users	Accessed using domain-level users



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

In the MT reference guides, REST resources:

- Are running in a partition
- Must be accessed over a partitioned URL by a user defined in that partition's security realm
- Can only be used to manage that partition
- Cannot be used to manage all WLS MBeans. Many of the WLS MBeans are not available to partition users.

In the non-MT reference guides, REST resources:

- Run at the domain level (versus in a partition)
- Must be accessed over a nonpartitioned URL by a user defined in the domain's default security realm
- Can be used to manage all partitions
- Can be used to manage all WLS MBeans

Supported Representations

WebLogic Server supports multiple representations:

- Extensible Markup Language (XML)

```
<person><age>30</age><name>chris</name></person>
```

Note that JSON is the preferred notation
- JavaScript Object Notation (JSON): A lightweight data-interchange based on name–value pairs

```
{ "name": "chris", "age": "30" }
```
- All management and edit interfaces use JSON exclusively
- HTML: A webpage written to represent a particular request limited to REST access via a built-in application.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic Server supports several representation with processing REST requests, including XML, JSON, and HTML.

- XML: There is nothing unusual or unexpected about WebLogic XML. Requests using XML are represented using “Accept:application/xml”.
- JavaScript Object Notation or JSON: A simple, lightweight data interchange format based on name–value pairs. JSON is fairly human readable, relatively easy to generate and parse, and flexible in terms of what it can represent. In its simplest form, JSON simply relates a name to a value. However, a value can be another object (represented by nested braces), an array, a null value, a true/false value, a string, or a number. Each of these types is well defined.
Requests using JSON are represented using “Accept:application/json”.
- HTML: Requests formatted as traditional HTML and including a body, an item, and some number of attributes

Note that the bulk of the management interfaces exclusively use JSON that is they cannot accept nor return XML or HTML.

For more information about JSON, see <http://www.json.org>.

REST Resource Access Roles

REST Resources:

- Are protected resources
- Can only be accessed via specific roles
- To access REST resources you must be one of:
Admin, Deployer, Operator, or Monitor roles

In general:

To perform	Require Role
GET	Read - Admin, Deployer, Operator, or Monitor
POST (add/update/delete /invoke)	Add, Update, Delete, Invoke - Admin
POST (add/delete/invoke)	With certain resources, a Deployer can deploy and undeploy applications and libraries, and an Operator can start a server.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

REST resource themselves define which user roles are required for each operation. The roles: Admin, Deployer, Operator, and Monitor are required for accessing REST resources.

In general:

- You must be in the Admin, Deployer, Operator, or Monitor role to read resources (use the GET method)
- You must be in the Admin role to write resources (use the POST and DELETE methods) or to invoke operations (using POST)

With certain resources, a Deployer can deploy and undeploy applications and libraries, and an Operator can start a server.

If the user is at the domain level, the URL to access REST resources starts with **http://host:port/management**. If the user is a partition user (for example, defined in that partition's security realm), the URL to access REST resources starts with **http://host:port/partition_name/management**.

For more information see: <https://docs.oracle.com/middleware/1221/wls/WLRUR/using.htm>

Agenda

- WebLogic Server and REST
- Working with WebLogic REST
- WebLogic Server REST Operations and Workflows



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic RESTful Interfaces

WebLogic Server REST URLs are of the form:



where:

- host: The host where WebLogic Server is running
- port: The HTTP or HTTPS port
- path: The relative path to a specific resource

```
curl -v --user username:password \
      -H X-Requested-By:SomeClient \
      -H Accept:application/json \
      -X GET \
      http://localhost:7001/management/weblogic/latest/servers
```

Several possible relative paths exist.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

To interact with WebLogic Server using URL, you must first understand the URL that represents the path to the resource. In all cases, the URL begins with a **http(s)** followed by the **host** name and **port** of the Administration server instance. Following the root portion of the URL is always the path **management**. The remainder of the path varies based on the source of the data and the operation being performed.

There are a number of paths which map directly to the WebLogic Bean tree. For example:

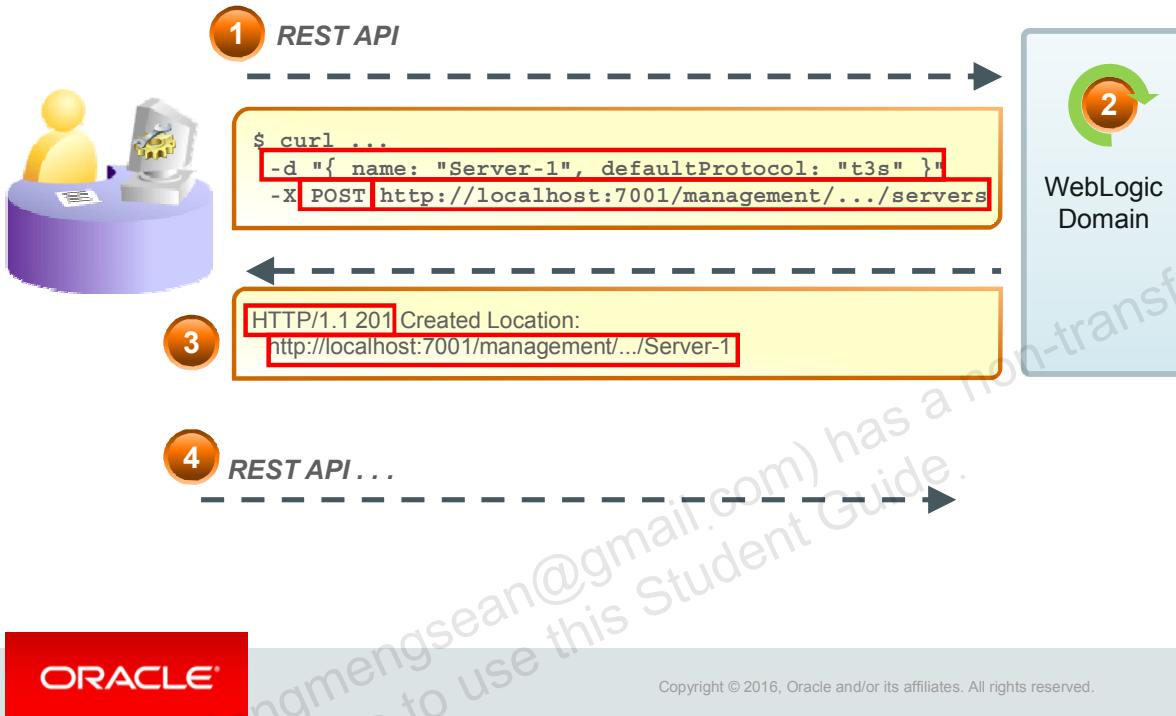
- **weblogic/{version}**: Represents the root of the entire WebLogic Bean tree. **{version}** can be either **latest**, or a specific supported version such as **12.2.1.0**.
- **weblogic/latest/**: Represents the current latest version of WebLogic and provides information on the root of the WLS Bean Tree, included **edit**, **domainConfig**, **domainRuntime**, **serverConfig**, **serverRuntime** and others

You will examine several of these examples later in the lesson.

Note that there are simple monitoring operations based on the tenant-monitoring path which are deprecated in the current release.

REST Operations Flow

REST operations follow a standard flow:



The process for performing WebLogic REST request is relatively straightforward. In general the following occurs:

1. A client fabricates and makes a REST request against a WebLogic domain. Depending on the request, an example is shown using cURL, different elements may be provided. For example, as shown in this example, a JSON payload may accompany the request and provide information for object creation or updates.
The request always includes the URL of the target object.
2. The WebLogic Domain then consumes the request, returning a response code as well as a JSON result.
3. The client then consumes the payload as appropriate.
4. Depending on the process flow, additional REST requests may or may not occur.

Example: Returning Server Information Using cURL

Returning server information:

```
$ curl --user username:password \
-H X-Requested-By:SomeClient \
-H Accept:application/json \
-X GET \
http://localhost:7001/management/wls/latest/servers
```

Result:

```
{"body":  
  {"items": [  
    {"name": "AdminServer",  
     "state": "RUNNING",  
     "health": "HEALTH_OK"},  
    {"name": "server1",  
     "state": "RUNNING",  
     "health": "HEALTH_OK"},  
    {"name": "server2",  
     "state": "SHUTDOWN"}  
  ],  
  "messages": []}
```

```
{"body":  
  {"item": {  
    "name": "server1",  
    "state": "RUNNING",  
    "health": "HEALTH_OK",  
    "clusterName": "cluster1",  
    "currentMachine": "machine1",  
    . . .  
    "javaVersion": "1.8.0_60",  
    "heapSizeCurrent": 334495744,  
    "heapFreeCurrent": 230418152},  
  "messages": []}}
```

Append /server1
to get this result.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

In the example given, cURL is used to access the localhost using a specific username and password. The request is specified as a GET operation via **-X GET** and the request results in json using **Accept:application/json**. The specific resource is the server list which is specified via a **wls/latest/servers** request.

The result is a list of three items including the name, state, and for running server instances health status.

The second example appends one of the running server names to the request. Note that some fields are omitted for space.

Typical REST Operations

Operation	Description
GET	Return data for the specified object
POST (insert)	Add a new object to the tree at the specified location
POST (update)	Update an existing object specified at the current location in the tree
POST (invoke)	Execute an operation on a object at the current location in the tree
DELETE	Delete an object from the tree



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic Server supports a number of REST requests including:

- GET: Return information for a specific object located at a given location based on its URL
- POST (insert): Insert the specific object, defined in a JSON payload, into the tree at the location given by the provided URL
- POST (update): Update the specific object, defined in a JSON payload, into the tree at the location given by the provided URL
- POST (invoke): Perform the given operation based on the URL+ operation. For example **http://hostname:port/management/wls/latest/servers/id/server1/start**, where **start** is the operation.
- DELETE: Delete the object specified by the URL
- OPTIONS: Note that OPTIONS is deprecated in 12.2.1 and only limited support is provided; use createFormXXX instead.

Return a JSON description of an object as specified by the URL. For example:
http://hostname:port/management/wls/latest/servers/id/datasources:

Common WebLogic REST Results

Return code	Description
200 (OK)	Success of the GET, OPTIONS or POST
201 (CREATED)	Return data for the specified object
202 (ACCEPTED)	
400 (BAD REQUEST)	Bad Request
401 (UNAUTHORIZED)	Unauthorized (bad credentials)
403 (FORBIDDEN)	User does not have the rights to perform operation.
404 (NOT FOUND)	Not found, likely invalid URI to resource that does not exist
406 (NOT ACCEPTED)	Not acceptable. Request cannot be accepted, for example asking for XML where only JSON is supported.
500 (INTERNAL ERROR)	The server is currently down.
503 (UNAVAILABLE)	REST isn't enabled.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

- 200 OK: The request was successfully completed. A 200 is returned for successful GET or OPTIONS methods, or a POST method to modify a resource (versus create).
- 201 Created: The request has been fulfilled and resulted in a new resource being created. A Location header containing the canonical URI for the newly created resource will be returned. A 201 is returned from a synchronous resource creation or an asynchronous resource creation that completed before the response was returned.
- 202 Accepted: The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, because it might be disallowed when processing actually takes place.
- 400 Bad Request: The request could not be processed because it contains missing or invalid information (such as a validation error on an input field, a missing required value, or such).
- 401 Unauthorized: The request is not authorized. The authentication credentials included with this request are missing or invalid.
- 403 Forbidden: Cannot authenticate the user. The user does not possess authorization to perform this request.

- 404 Not Found: The request specified a URI of a resource that does not exist.
- 405 Method Not Allowed: The HTTP verb specified in the request (DELETE, GET, HEAD, POST, PUT) is not supported for this request URI.
- 406 Not Acceptable: The resource identified by this request is not capable of generating a representation corresponding to one of the media types in the Accept header of the request. For example, the client's Accept header asks for XML but the resource can only return JSON.
- 415 Not Acceptable: The client's ContentType header is wrong (for example, the client tries to send in XML but the resource can only accept JSON).
- 500 Internal Server Error: The server encountered an unexpected condition which prevented it from fulfilling the request.
- 503 Service Unavailable: The server is currently unable to handle the request due to temporary overloading or maintenance of the server. The WLS REST web application is not currently running.

WebLogic REST Error Handling

WebLogic REST errors include:

- **type**: Type of the error
- **title**: Error title *Returned for single errors only*
- **detail**: Message representing details *←*
- **status**: Status code *Array when more than one error occurred*
- **wls:errorsDetails**: Array of error messages *←*

```
HTTP/1.1 400 Bad Request {  
    type: "message type information",  
    title: "Single Error",  
    detail: "Some error occurred, error details",  
    status: 400 }
```

```
HTTP/1.1 400 Bad Request {...  
    title: "Multiple Errors", ...  
    wls:errorsDetails: [  
        { type: "...", title: "1st Error", detail: "n...", o:errorPat: "field or other informaiton" },  
        {...} ]  
    ]}
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

As with any API, WebLogic REST can return errors. On addition to the HTTP status code, an error structure is provided that includes error details. Details include:

- **type**: String, error typing information. See WebLogic REST error documentation for more information on type support.
- **title**: String, short error title
- **status** : Integer, error code
- **detail**: String, present if single error only
- **wls:errorDetails**: An array of **type**, **title**, **detail**, and **o:errorPat** elements. Only present when multiple errors occur.

Understanding WebLogic REST Requests

WebLogic REST request relative paths are composed of:

- Parent reference: Reference to a location in the WebLogic MBean hierarchy

```
.../management/weblogic/<version>/edit/...
.../management/weblogic/<version>/serverConfig/... Legacy root
.../management/wls/<version>/jobs/...
```

- Resource reference: Path to a specific resource root, collection or named resource beneath a parent

```
.../<collection>           (.../edit/servers )
.../<collection>/<child>    (.../edit/servers/server1)
```

Examples:

```
.../management/weblogic/latest/edit
.../management/weblogic/latest/edit/servers
.../management/weblogic/latest/edit/servers/server1
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

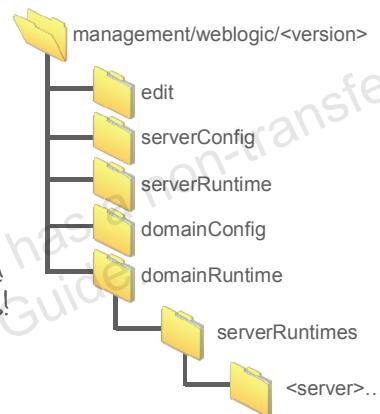
WebLogic REST use a very specific format for requests. Each request contains basic information such as `http://hostname:port` and possibly a partition-specific identifier. Beyond these core elements, references are based on:

- Parent: The parent portion of the URL specifies where in the WebLogic bean tree the reference should begin. Some examples for editing the domain configuration, accessing a running server configuration, and examining a `serverRuntime` are shown. Beyond the parent references are:
- Collection: One of a number of collections such as **servers**, **datasources**, and others. Collections are generic in the sense that they always exist. Beneath collections are:
- Child – Specific elements within a collection by name. For example **server1** might be the name of a child resource within a the **servers** parent beneath the **edit** root.

WebLogic REST MBean Hierarchy

- Is accessible via `/management/weblogic/<version>`
- Is used extensively by WebLogic components
- Can manage configuration settings
- Can monitor and manage running servers
- Are constructed at run time from WebLogic Mbeans
- Support:
 - Configuration
 - Runtime (Monitoring and management)

Can be used in a similar fashion to WLST to interact with, manage and monitor WebLogic Server instances!



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic Server supports what are collectively referred to as WebLogic MBeans. MBeans are used extensively by WebLogic Server itself for configuration and runtime support. These MBeans contain runtime information as well as configuration and settings. Many are generated and populated at run time. WebLogic Scripting Tool or WLST makes extensive use of the MBean tree to provide data to manage and configure WebLogic Server. Likewise, WebLogic REST exposes the MBean tree under the root `/management/weblogic/<version>`. As with WLST, the tree is broken down into 5 major categories or areas:

- **edit**: Underlying edit access bean tree
- **domainConfig** : Configuration MBean half of the domain access bean tree (such as, the last persisted configuration)
- **domainRuntime**: Runtime MBean half of the domain access bean tree (such as, for monitoring all servers)
- **serverConfig**: Configuration MBean half of the runtime access bean tree (such as, the configuration the server is using)
- **serverRuntime**: Runtime MBean half of the runtime access bean tree (such as, for monitoring a specific server)

In addition, the domain runtime branch of the **weblogic** tree contains a **serverRuntimes** with specific server children.

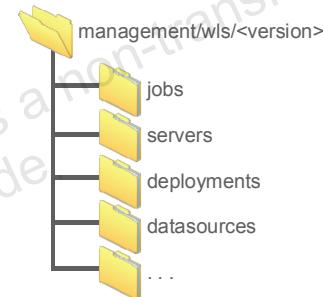
WebLogic Legacy REST Resources

WebLogic Legacy Resources:

- Are accessible via `/management/wls/<version>/`
- Provide a legacy view of WebLogic Server
- Support access servers, jobs, deployments and other resources

```
$ curl --user username:password \
      -H Accept:application/json \
      -X GET http://localhost:7001/management/wls/latest/
```

```
{
  "item": {
    "activeHttpSessionCount": 0,
    . . .
    "name": "wlsadmin"
  },
  "links": [
    {
      "rel": "parent",
      "uri": http://localhost:7001/management/wls"
    },
    . .
  ]
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic Server also supports what's known as legacy references to resources using a slightly shortened path of `/management/wls/<version>`. The legacy tree is a small subset of the entire MBeans tree and includes child elements such as jobs, servers, deployments, datasources and others. These legacy references look and behave as their MBean tree cousins do, but are slightly easier to find in the tree. The example shown lists all legacy references as links as well as a set of parent items.

From a historical perspective the WLS RESTful Management Interface has gone through several major designs:

- 12.1.1 and 12.1.2: The `tenant-monitoring` interface is introduced, with support for XML and JSON. Deprecated as of 12.1.3.
- 12.1.3: The `wls` interface is introduced, with JSON support for specific creation, deployment, monitoring and lifecycle actions. Referred to as legacy in current documentation.
- 12.2.1: The `management/weblogic` interface is introduced, with full MBean tree access and support for Multitenancy.

WebLogic REST Response Bodies

WebLogic REST Responses contain:

- Items: An array of all items based on the request
- Identify: An item within the items array uniquely identifying the requested resource
- Properties: A set of properties for the given request
- Links: A set of links representing related elements

```
{
  "identity": [
    "servers",
    "server1"
  ], ...
}
```

```
{
  "identity":...
  "type": "Server",...
  "cluster": [ "clusters", "cluster1"],...
  "machine": [ "machines", "machine1"],...
}
```

```
{
  "links": [
    { "rel": "parent",
      "href": "http://localhost:7001/.../domainConfig/servers" },
    { "rel": "cluster",
      "href": " http://localhost:7001/.../domainConfig/cluster1" },
    ... ] }
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic REST requests, while not 100% consistent across all requests, return a common set of elements including:

- Identity: The **identity** element uniquely describes the result and provides both the parent element and the name of the child element. For a server request, such a result would be **servers** and **server1** (parent/child).
- Properties: A set of resource-specific properties. Each resource property is mapped to the appropriate JSON type. See the WebLogic Server REST overview for details on how specific types are mapped from Java to their JSON equivalent.
- Links: The **links** element is an array of all the related links associated with the resource. Typically all resources return a **parent** element and a **self** element as well as other elements specific to the type. For example, a server instance might contain a **cluster** link, representing the cluster the server is a part of, a **machine** link, representing the machine the server is managed under, and other similar links. By adding the **?links=none**, the links portion can be removed from the returned result.

Note that the legacy REST APIs return a variation of the fields described here. See the WebLogic Server REST legacy documentation if you want to use legacy resources. Remember that legacy resources may be deprecated in a future version of WebLogic Server.

Example: Listing Server Information by Name

```
$ curl --user username:password \  
  -H Accept:application/json \  
  -X GET http://localhost:7001/management/wls/latest/servers/id/server1
```

```
{"item": {  
    "heapFreeCurrent": 193964720,  
    "heapSizeCurrent": 331874304,  
    . . .  
    "health": {"state": "ok"},  
    . . .  
    "name": "server1",  
    "state": "running"  
},  
"links": [  
    { . . . },  
    {  
        "rel": "logs",  
        "uri": "http://localhost:7001/. . . /latest/servers/id/server1/logs"  
    }  
]
```

Details of server1

Associated links



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

In the example shown, a request was made to a specific host for the URL /management/wls/latest/servers/id/server1.

Invoking Operations

Resources may support operations.

Resource operations:

- Map to a distinct child URL ↗ .../serverRuntimes/Server1/shutdown
- Uses the same URL for all methods of the same name
- Support input arguments via JSON request body objects
- Uses input arguments to disambiguate overloaded methods
- Return results via response bodies
- Can throw exceptions as 404 errors

```
$curl ... -d { timeout: 500, ignoreSessions: false } \
-X POST http://localhost:7001/management/weblogic/latest/
domainRuntime/serverRuntimes/Server-1/shutdown
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Many WebLogic Server MBeans support operations. For example servers might support, start, stop, suspend, shutdown and so on. For those MBeans which do support operations:

- Each method maps to a child URL of the parent resource
- All methods of the same name use the same name with overloaded methods disambiguated by parameters
- Support parameters via named parameter pairs in JSON request body objects
- Return results via response bodies, void methods returning nothing
- Can throw exceptions which are returned as 404 errors along with error text in the body of the response

Create Forms

Create forms

- Are a mechanism for obtaining resource properties
- Are typically used when creating or updating resources
- Are named based on the resource served
- Are available in the edit tree

```
$ curl ... \
  -H Accept:application/json \
  -X GET \
  http://localhost:7001/management/weblogic/latest/edit/clustercreateForm
```

```
{
  "links": [ . . . ],
  "sessionStateQueryRequestTimeout": 30,
  "notes": null,
  "sessionFlushInterval": 180,
  . . .
  "defaultLoadAlgorithm": "round-robin",
  . . .
  "name":null
}
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic resources can have a very large number of associated properties. Not all requests return all properties. As such a simple GET of an existing resource is rarely sufficient to list properties for that type of object. Create forms solve this problem by providing a mechanism to list all properties for a give resource. Typically create forms exist in edit tree at the same relative location as the resource in question. For example, server, cluster and machine create forms exist as children of the edit root because they have no higher parent. To obtain the properties for a given resource use the type name a prefix for the CreateForm request. For example, for machines use a request similar to:

```
curl -v --user weblogic:Welcome1 \
  -H X-Requested-By:MyClient \
  -H Accept:application/json \
  -X GET
http://localhost:7001/management/weblogic/latest/edit/machinecreateForm
```

Which would produce a result similar to:

```
{  
  "links": [  
    {  
      "rel": "parent",  
      "href": "http://localhost:7001/management/weblogic/latest/edit"  
    },  
    {  
      "rel": "self",  
      "href":  
      "http://localhost:7001/management/weblogic/latest/edit/machinecreateForm"  
    },  
    {  
      "rel": "canonical",  
      "href":  
      "http://localhost:7001/management/weblogic/latest/edit/machinecreateForm"  
    },  
    {  
      "rel": "create",  
      "href":  
      "http://localhost:7001/management/weblogic/latest/edit/machines"  
    }  
  ],  
  "notes": null,  
  "tags": null,  
  "name": null  
}
```

Creating Resources

Resources are created using:

- JSON request bodies: Containing property name:value pairs
- POST operations against the parent resource

```
$ curl ... \
  -H Content-Type:application/json \
  -d " { name: \"server3\", listenPort:\"7013\",
        defaultProtocol:\"t3s\",
        \"cluster\": [ \"clusters\", \"cluster1\"],
        \"machine\": [ \"machines\", \"machine1\" ] }" \
  -X POST http://host01:7001/management/weblogic/latest/edit/servers
```

- Use serverCreateForm to list known properties

```
$ curl ... \
  -H Accept:application/json \
  -X GET \
  http://localhost:7001/management/weblogic/latest/edit/servercreateForm
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

In addition to returning information about known existing resources, WebLogic REST can be used to create new resources. The example shown below creates a new server resource:

```
$ curl --user username:password \
  -H Content-Type:application/json \
  -d " { name: \"serverTest\",
        listenPort:\"7013\",
        defaultProtocol:\"t3s\",
        \"cluster\": [ \"clusters\", \"cluster1\"],
        \"machine\": [ \"machines\", \"machine1\" ] }" \
  -X POST
http://host01:7001/management/weblogic/latest/edit/servers
```

Note that several lines have been split for readability.

Agenda

- WebLogic Server and REST
- Accessing WLS using REST
- WebLogic Server REST Operations and Workflows



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Common Workflows

Modify a configuration using an edit

```
# Start edit
$ curl ... -H Content-Type:application/json \ -d "{}" \
-X POST http://localhost:7001/management/weblogic/latest/edit/changeManager/startEdit

# get the cluster create Form (to determine available fields)
$ curl ... -H Accept:application/json \
-X GET
http://localhost:7001/management/weblogic/latest/edit/clustercreateForm?links=none

# Add the cluster
$ curl ... -H Accept:application/json \
-H Content-Type:application/json \
-d "{ name: 'MyCluster' }" \
-X POST http://localhost:7001/management/weblogic/latest/edit/clusters

# Activate changes
$ curl ... -H X-Requested-By:MyClient \
-H Accept:application/json \
-H Content-Type:application/json \ -d "{}" \
-X POST http://localhost:7001/management/weblogic/latest/edit/changeManager/activate
```

Select the methods with no parameters.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Using edit is analogous to clicking the edit button in the WebLogic Server console. This example shows:

- Start the edit: Use the changeManager/startEdit method to begin an edit.
- Use the clusterCreateForm request to list all the fields available for use with servers; do not report links.
- Add a new cluster.
- User the changeManager/activate methods to activate the change.

For example, multiple changes could be grouped together in this fashion. Adding a cluster, then adding machines to the cluster, then adding servers using the cluster and machine then activating all the changes as a single transaction.

Common Workflows

Deploy/View/Un deploy an application

```
$ curl --user someDeployer:password \
-H Accept:application/json \
-H Content-Type:application/json \
-d "{
  name:      'myApp',
  sourcePath: '/fully/qualified/path/to/warfile/myApplication.war',
  targets:    [ { identity: [ 'clusters', 'clusterName' ] } ]
}" \
-X POST http://localhost:7001/management/weblogic/latest/edit/appDeployments
```

Application name,
fully qualified path to
war and target

Add

```
$ curl . . . \
-H Accept:application/json \
-X GET http://localhost:7001/management/weblogic/latest/edit/appDeployments/MyApp
```

View

```
$ curl . . . \
-H Accept:application/json \
-X DELETE http://localhost:7001/management/weblogic/latest/edit/appDeployments/MyApp
```

Delete



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

To deploy an application use a **POST** request again, the **management/weblogic/latest/edit/appDeployments** resource. Provide, as a JSON body, the application name, fully qualified path to the application and a target, in the example case the cluster **clusterName**. To delete an application, use a **DELETE** request against the **appDeployment** root providing the registered name of the application. Likewise, to view an application use a **GET** request again, the **appDeployment** root, providing the registered application name.

Common Workflows

Start/View/Stop Servers

```
$ curl . . . \
-H Accept:application/json \
-X GET http://localhost:7001/management/weblogic/latest/
domainRuntime/serverLifeCycleRuntimes?links=none&fields=name,state
```

List all running servers.

View

```
$ curl . . . \
-H Accept:application/json \
-H Content-Type:application/json \
-d "{ timeout: 10, ignoreSessions: true} " \
-X POST http://localhost:7001/management/weblogic/latest/
domainRuntime/serverLifeCycleRuntimes/SomeServer/shutdown
```

Do not return links and limit fields.

Stop

Execute shutdown operation with provided parameters.

serverLifeCycleRuntimes used to manage servers

```
$ curl . . . \
-H Accept:application/json \
-H Content-Type:application/json \
-d "{}" \
-X POST http://localhost:7001/management/weblogic/latest/
domainRuntime/serverLifeCycleRuntimes/SomeServer/start
```

Start



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

To view running servers, use the **domainRuntime/serverLifeCycleRuntimes** path. Note the use of **links=none** and **fields=fieldList** to limit the values returned.

To stop a running server, use a **POST** operation against the servers name specifying the **shutdown** operation. Force shutdown and other methods are also available.

To start a server, use a **POST** operation against the server name using the **start** command. Note that a node manager instance must be up and running on the machine where the server should run.

Common Workflows

Monitor Servers

```
$ curl . . . \
-H Accept:application/json \
-X GET http://localhost:7001/management/weblogic/latest/
domainRuntime/serverRuntimes
?links=none&fields=name,overallHealthState,healthState,state
```

Monitor all running servers.

Ignore links, and return select fields.

```
{
  "items": [
    {
      "overallHealthState": { "state": "ok", "subsystemName": null,
        "partitionName": null,
        "symptoms": [] },
      "healthState": { "state": "ok", "subsystemName": null, "partitionName": null,
        "symptoms": [] },
      "name": "Server1",
      "state": "RUNNING"
    } ,
    .
    .
    .
    "name": "ServerX",
    "state": ". . . "
  } .
  ]
}
```

Return an array of server information based on the selected fields.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

To monitor running servers, use the **domainRuntime/serverRuntimes** path. Note the use of **links=none** and **fields=fieldList** to limit the values returned.

Summary

In this lesson, you should have learned how to:

- Describe WebLogic REST capabilities
- Create and execute WebLogic REST requests
- Evaluate WebLogic REST results and responses
- Monitor, manage, and configure WebLogic using REST



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Quiz



WebLogic REST Create forms:

- a. Are a mechanism for obtaining resource property templates
- b. Are used when updating, or creating resources
- c. Are only used when updating resources
- d. Are named based on the resource served
- e. Are exclusively in the legacy tree
- f. Can return JSON, XML, or HTML formats

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: a, b, d

Create forms are a mechanism for obtaining JSON examples of response bodies, often used as templates. They are used to return a complete set of all properties associated with a given request. Create forms are obtained using requests of the form **edit/xxxCreateForm** where xxx is one of supported resource types such as cluster, server or many others. Create forms are available exclusively in the edit tree and can return JSON only.

Practice 7-1 Overview: Creating and Managing Servers Using REST

This practice covers the following topics:

- Confirming that a domain is configured to support REST
- Creating a server in a known domain using REST and JSON
- Starting a server using REST
- Listing various information about servers using REST



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic REST Monitoring Relative Paths

Monitoring supports the following relative paths:

Path (/tenant-monitoring)	Description
/servers [/{{instance}}]	Returns information for servers configured in a domain or for a specific server in a domain
/clusters [/{{instance}}]	Returns information for all clusters configured in a domain or for a specific cluster in a domain
/datasources [/{{instance}}]	Returns information for all data sources configured in a domain or a specific data source
/applications [/{{instance}}]	Returns information for all deployed applications or a specific application

These operations are typically only used via GETs.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic Server supports monitoring of domains, server, applications, and data sources using the tenant-monitoring root. Each resource can be accessed either as a list or as a single instance by appending the name of the resource to the path. For example to see details of a specific server, append its name to the root such as **/tenant-monitoring/servers/AdminServer**. Typically, the parent request, servers vs a specific server, returns general information about the set of artifacts where the instance request returns details.

See Administering Oracle WebLogic Server with RESTful Management Services for complete details on what is required for each call.

Note that the tenant-monitoring interface is being deprecated and will be removed in future released.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Sean Tay (taysiangmengsean@gmail.com) has a non-transferable
license to use this Student Guide.

Secure Sockets Layer (SSL)

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe SSL concepts
- Configure keystores in WebLogic
- Configure SSL for WebLogic architectures



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Agenda

- SSL Concepts
 - What Is SSL?
 - SSL Terminology
 - Symmetric and Asymmetric Encryption and Decryption
 - Digital Certificates and Certificate Authorities
 - SSL Communication
- WebLogic SSL Scenarios
- Keystores
- Configuring WebLogic SSL

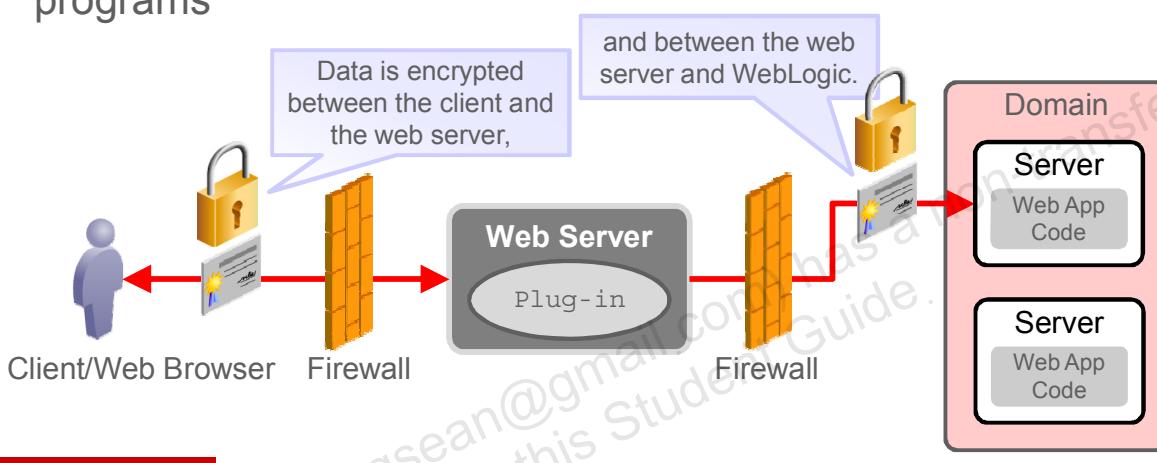


Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

What Is SSL?

SSL stands for Secure Sockets Layer, which is used primarily for two purposes:

- To verify the identity of a website and, optionally, client identities
- To establish a secure encrypted connection between programs



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The SSL protocol offers security to applications that are connected through a network. Specifically, the SSL protocol provides the following:

- Encryption of the data that is exchanged by applications
- A mechanism that the applications can use to authenticate each other's identity
- Data integrity, whereby the data that flows between a client and a server is protected from tampering by a third party

When the SSL protocol is used, the target always authenticates itself to the initiator.

Optionally, if the target requests it, the initiator can authenticate itself to the target. Encryption makes the data that is transmitted over the network intelligible only to the intended recipient. An SSL connection begins with a handshake during which time the applications exchange digital certificates, agree on the encryption algorithms to be used, and generate the encryption keys to be used for the remainder of the session.

Note: The only SSL stack supported by Oracle WebLogic Server 12c is the Java Secure Socket Extension (JSSE) stack. The Certicom-based SSL implementation is removed and no longer supported.

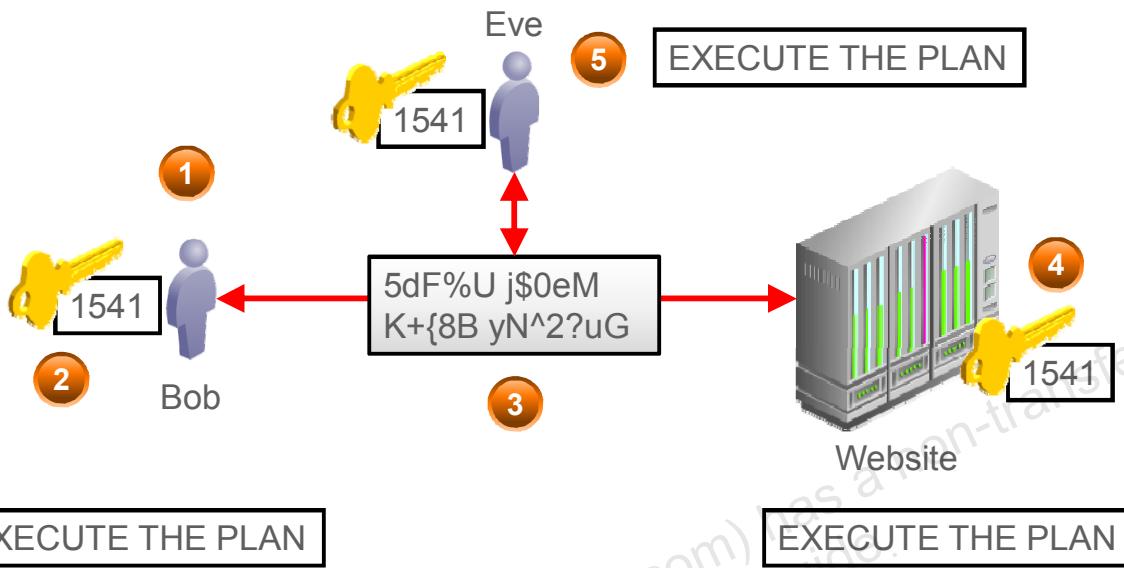
SSL Terminology

Term	Description
Encryption	A process used to obfuscate data so it is unreadable by anyone except the intended party
Decryption	A process used to revert encrypted data back to its readable form
Digital signature	A process of providing a hashed version of data, which is secured using encryption, to establish the trustworthiness of that data
Key	A series of numbers used to perform the encryption and decryption processes
Symmetric key	A single key that is used to encrypt and decrypt messages
Asymmetric key	A pair of different keys that are mathematically related. This consists of a public key that is shared and a private key that is kept secret.
Public Key Infrastructure (PKI)	A security infrastructure and process that uses all of these artifacts to establish secure communication between parties



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Symmetric Encryption and Decryption

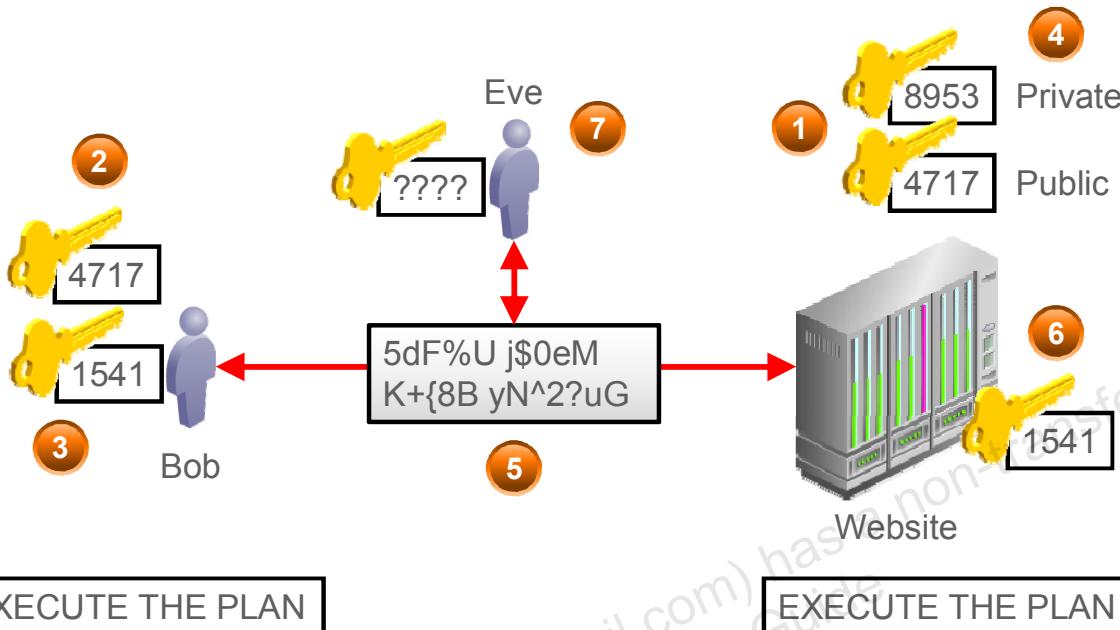


ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

1. Bob wants to send private data to a website. He wants to use a symmetric key, 1541, to encrypt his data. He sends the key to the website servers so they will be able to decrypt his data.
2. Bob uses his key to encrypt the data, "EXECUTE THE PLAN".
3. Bob sends the encrypted form of his data to the website.
4. The website receives Bob's data and uses his key to decrypt the data. The website then sees that his message is, "EXECUTE THE PLAN".
5. Unfortunately, Eve the eavesdropper was watching when Bob gave his key to the website. Therefore, she is able to decrypt Bob's message and also sees that his message is, "EXECUTE THE PLAN".

Asymmetric Encryption and Decryption



1. Once again, Bob wants to send private data to a website. Bob finds that the website offers asymmetric keys for exchanging the symmetric key used for encrypting and decrypting data.
2. The website gives Bob its public key. Bob knows that data encrypted with this key can be decrypted only by the associated private key, and that only the website has that private key.
3. Again, Bob wants to use a symmetric key, 1541, to encrypt his data. This time, he uses the website's public key to encrypt what his symmetric key is, and then he sends the encrypted key to the website so they will be able to decrypt his data.
4. The website receives Bob's key and uses its private key to decrypt Bob's symmetric key.
5. Bob uses his symmetric 1541 key to encrypt the data, "EXECUTE THE PLAN" and sends the encrypted form of his data to the website.
6. The website receives Bob's data and uses his key to decrypt the data. The website then sees that his message is, "EXECUTE THE PLAN".

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

7. And again, Eve the eavesdropper was watching when Bob gave his key to the website. However, she could not figure out what Bob's symmetric key was. Eve is not able to decrypt and read Bob's message. The only problem here is, how does the website know that the message came from Bob? What is stopping Eve from pretending to be Bob and send messages of her own? Or for that matter, how does Bob know that the public key he got from the website was not actually sent by Eve and he just freely gave her his symmetric key and private data?

Digital Certificates

Digital certificates are electronic documents used to verify identities and entities over networks:

- They provide digital encryption and signature artifacts used to make SSL communication possible.
- The most widely accepted standard is X.509.
- They are used to establish trust that the holding party is who they say they are.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Digital certificates are used to establish the identity of the holder of the certificate. Digital certificates contain the public key of the holder as well as the identity of the issuer of the certificate. The issuer of the certificate in turn has its own certificate, which is used to establish the trust required to validate the identity of the original certificate. The issuer's certificate can also be issued by another party that is also backed by a certificate. The concept of one certificate leading to another in sequence is called a certificate chain. A root certificate is found at the end of the chain. The root certificate is a self-signed certificate of a well-known and trusted company, such as VeriSign. A self-signed certificate is one for which the subject (identity) of the certificate is the same as the issuer of the certificate.

Digital Certificate: Example

```
openssl x509 -in mycert.der -inform DER -noout -text
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number: 660557372 (0x275f4e3c)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: CN=wls-sysadm
    Validity
        Not Before: Feb 20 13:35:19 2013 GMT
        Not After : Feb 20 13:35:19 2014 GMT
    Subject: CN=wls-sysadm
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        Public-Key: (512 bit)
        Modulus:
            00:df:cc:8a:fe:91:50:6d:80:56:7d:f1:d0:d4:f1:
            c6:e3:f3:6f:cc:22:39:2d:ef:ac:7c:6c:4b:56:81:
            1c:de:d0:7d:ad:ea:47:63:98:15:74:37:66:dd:a1:
            64:09:2e:56:ba:65:c1:b8:15:55:ac:b5:09:fe:d6:
            32:0f:0c:7f:ff
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Subject Key Identifier:
        EA:4C:C3:08:B4:60:6A:B6:99:B4:15:70:E9:24:F2:18:97:46:E9:07
Signature Algorithm: md5WithRSAEncryption
3b:2f:0d:90:bb:82:dc:16:f2:8f:00:c3:40:d8:a5:3f:4b:39:
8f:52:27:b1:4d:e1:f4:85:f5:29:c8:ef:b4:99:06:ee:cd:96:
5c:6d:c8:a2:7b:1e:61:03:a1:c0:60:c8:90:b3:f9:ff:bf:82:
49:37:53:08:dd:c9:5f:ec:0b:a9
```



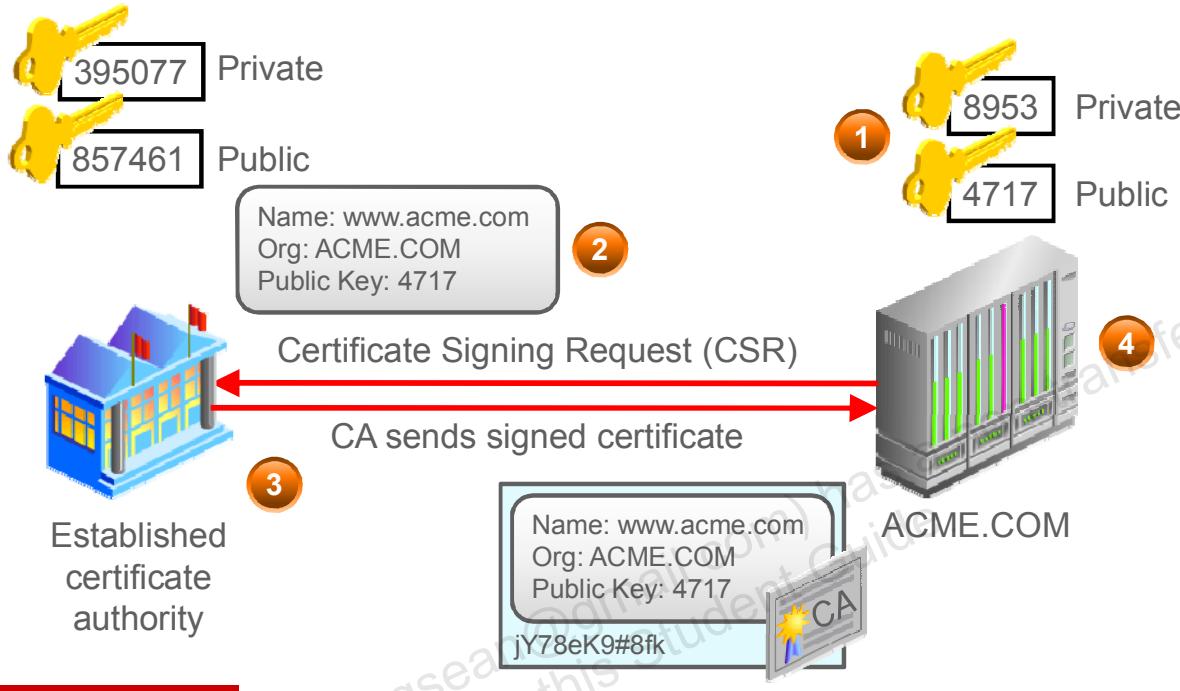
Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of a digital certificate that has been exported from a keystore in Distinguished Encoding Rules (DER) format. The certificate comprises several aspects that, when combined, provide all the required means of verifying a user's or an entity's identity and legitimacy:

- It specifies the algorithm used to create the digital signature that is used to verify the integrity of the message sent by the original sender.
- It specifies the issuer of the certificate. In this case, it is a self-signed certificate because the Issuer and the Subject are the same value. Normally, the certificate is issued by a trusted certificate authority that vouches for the trustworthiness of the subject or sender of the certificate. Self-signed certificates are useful for development or unit-testing purposes.
- It indicates the time frame for which the certificate and its data are valid.
- It specifies the identity, or subject, the certificate represents and the public encryption key and algorithm for the certificate. The corresponding private key used to decipher data encrypted or signed with the public key is stored on the subject (sender) side in a keystore.

Certificate Authorities

Digital certificates are issued by certificate authorities (CAs):



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

CAs are trusted, third-party organizations that are willing to vouch for the identities represented by the certificates they issue.

1. An administrator at ACME.COM generates a private key and matching digital certificate, and creates a CSR with the public key.
2. The administrator sends the CSR to a well-known CA to get an officially "vouched for" form of their key pair.
3. The CA receives the request and begins its process of verifying the identity of the CSR. After the CA has successfully validated the requester of the certificate, the CA digitally signs the certificate with the CA's private key to ensure the integrity of the certificate. This digital signature involves performing a hash of the certificate and encrypting the hash by using the CA's private key. The CA then sends the requested certificate to the administrator at ACME.COM.
4. The administrator receives the certificate and verifies the signature of the issuing CA by using the CA's public key. Now Bob can verify the identity of the certificate when he receives it from the website.

CA certificates can be signed by other CAs to form a certificate chain until ultimately the root certificate is reached, which is a self-signed certificate of a very well-trusted organization.

SSL Communication

There are two SSL approaches between a client and a server:

- One-way SSL
 - Enables a server to identify itself to the client
- Two-way SSL
 - Enables a server to identify itself to the client
 - Enables the client to identify itself to the server



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

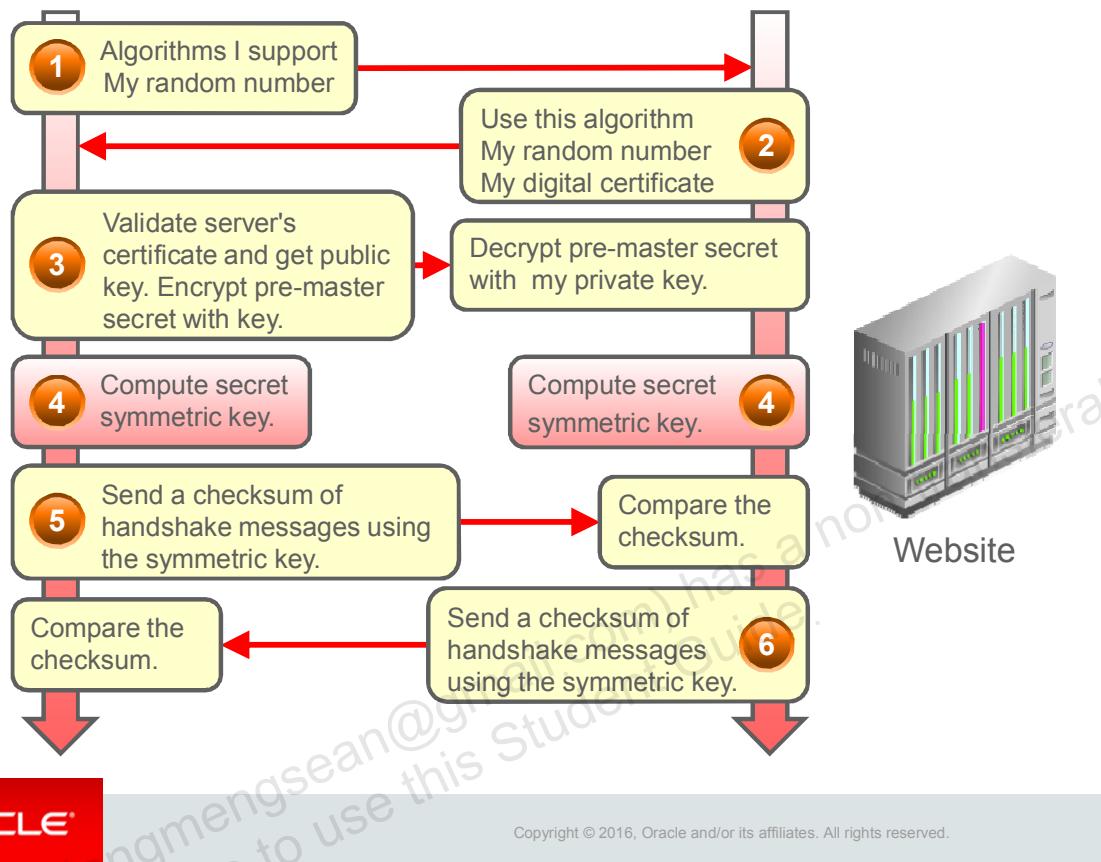
In a one-way SSL configuration, the application accessed by client browsers is an SSL server; the browsers that connect to the application over the Internet are SSL clients. The SSL client initiates contact with the SSL server. The SSL server presents a signed certificate (public key) to the SSL client. The SSL client verifies the identity of the server by following its certification chain.

In a two-way SSL configuration, the SSL client initiates a connection to an SSL server. Then the server presents its certificate to the client for verification and requests the client's certificate for verifying its identity.

One-Way SSL Handshake



Bob

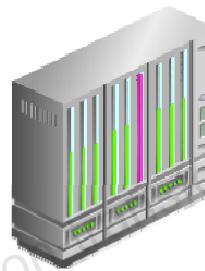
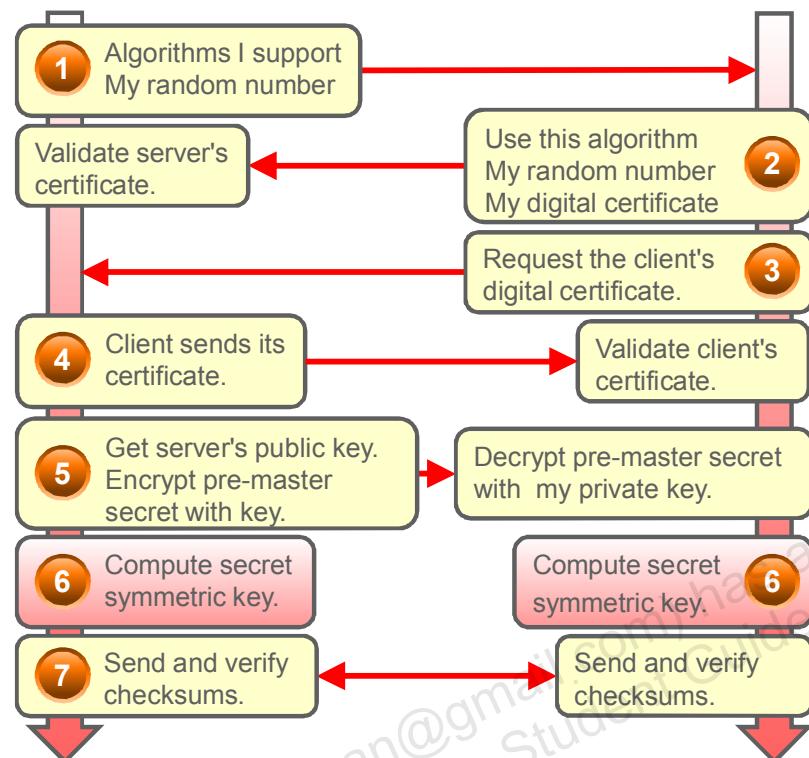


Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The steps in the slide negotiate the SSL session before data is passed between the two parties. The passing of checksum messages ensures that nobody tampered with handshake messages.

Two-Way SSL Handshake

Bob



Website

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The steps in the slide negotiate a two-way SSL session, which is very similar to the one-way steps, except that the client sends its certificate to the server and the server verifies the identity of the client. This is also used to authenticate clients by extracting the client's subject (identity) from its certificate.

Quiz



Digital certificates are issued by:

- a. Certificate domains
- b. Certificate committees
- c. Authorized certificate emitters
- d. Certificate authorities

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: d

Quiz



In SSL, _____ are used to provide a trusted public key.

- a. Certificate authorities
- b. Digital certificates
- c. JSSE sockets
- d. Encrypted files

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: b

Agenda

- SSL Concepts
- WebLogic SSL Scenarios
 - WebLogic and SSL
 - Demo Certificates
 - WebLogic SSL Requirements
 - WebLogic SSL Connections
- Keystores
- Configuring WebLogic SSL

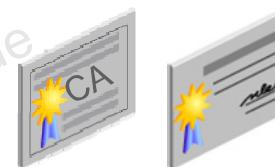


Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic and SSL

- WebLogic supports both one-way and two-way SSL.
- WebLogic uses SSL artifacts stored in keystores:
 - Identity:
 - Private key and digital certificate
 - Trust:
 - Digital certificates of trusted certificate authorities

More on keystores later



ORACLE®

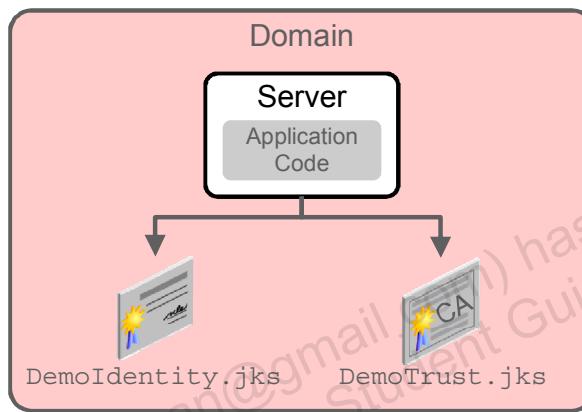
Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Oracle WebLogic Server supports the SSL protocol to enable secure communication between the applications that are connected over a network. By default, WebLogic Server is configured for one-way SSL authentication where servers are configured with digital certificates. Using the administration console, you can configure WebLogic for two-way SSL authentication where both the client and the server supply digital certificates to securely establish their identities.

Demo Certificates

Demonstration certificates are provided out-of-the-box for development:

- DemoIdentity.jks for identity
 - Separate demo certificates per domain
- DemoTrust.jks for trust



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic provides demo certificates for identity and trust out-of-the-box. All domains that are created get a demo certificate for easy development using SSL and other security features that rely on digital certificates. Whenever you create a new domain, the demo identity certificates are created and placed in demonstration keystores.

For demonstration purposes, you can use the following out-of-the-box:

- <DOMAIN_HOME>/security/DemoIdentity.jks: For identity
- <WL_HOME>/server/lib/DemoTrust.jks or <JAVA_HOME>/jre/lib/security/cacerts: For trust

By default, the Node Manager and server SSL use DemoTrust.jks for trust. A production environment should never use demonstration certificates because they are not trusted by a well-known CA.

Note: Demo identity certificates in previous releases of WebLogic were located in <WL_HOME>/server/lib. Now each domain has its own exclusive set of certificates. Demo trust certificates are still located in <WL_HOME>/server/lib.

WebLogic SSL Requirements

To enable SSL on WebLogic, you must perform the following steps:

1. Obtain an appropriate digital certificate.
2. Install the certificate.
3. Configure SSL properties.
4. Configure two-way authentication (optional).

Note: SSL impacts performance.



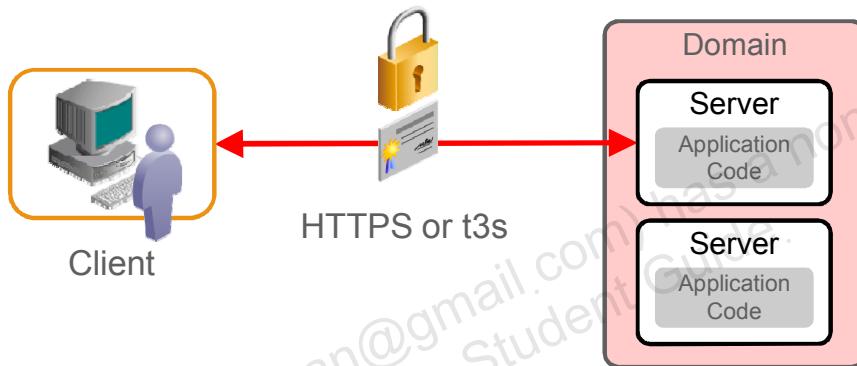
ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

There are several steps to configure WebLogic to use SSL. You must first obtain a valid certificate from a CA, such as VeriSign, Inc. You must then install the certificate, as well as the certificates of one or more certificate authorities that you trust. You can then optionally configure WebLogic to support two-way authentication by requiring clients to supply certificates. It is important, however, to remember that enabling security has a performance penalty. Packets need to be encrypted and tunneled over the network. Encryption and decryption are also CPU-intensive operations that can impact performance. However, when security is required, the performance penalty is usually worth it.

WebLogic SSL Connections

- WLS uses SSL to secure HTTP and t3 communication.
- To use SSL, clients access WLS via the HTTPS or t3s protocols.
 - `https://localhost:7002/orderStock`
 - `t3s://localhost:7002/useCreditCard`



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The use of SSL is signified in the protocol scheme of the URL used to specify the location of WebLogic servers. SSL communications between web browsers and WebLogic are transmitted in HTTPS packets (for example, `https://myserver.com:7002/mypage.html`).

WebLogic supports HTTPS with web browsers that support SSL version 3. Java clients connect to Oracle WebLogic Server with the SSL protocol tunnel over Oracle's multiplexed t3 protocol (for example, `t3s://myserver.com:7002`).

WebLogic Java clients can also establish either t3s connections to other WebLogic servers, or HTTPS connections to other servers that support the SSL protocol, such as web servers or secure proxy servers. Java clients and browsers connect securely to WebLogic by specifying the appropriate protocol (that is, HTTPS) in the requested URL. Clients can use JNDI to set up an SSL connection (for example, to an EJB). This is done by specifying a t3s connection within `PROVIDER_URL` and "strong" as the `SECURITY_AUTHENTICATION` type when populating the `Hashtable` object that is used to create the JNDI `InitialContext`.

Note: The `InitialContext` is coded by developers.

Oracle's JSSE implementation is the SSL implementation used by WebLogic. It is a collection of Java packages that allow for secure Internet communications. It is a Java implementation of the SSL and Transport Layer Security (TLS) protocols that allow for encryption, authentication (both server and client), and message integrity. After the client imports the proper packages and initializes the JSSE service, it uses the standard `java.net.HttpURLConnection` to create a secure connection.

Agenda

- SSL Concepts
- WebLogic SSL Scenarios
- Keystores
 - What Is a Keystore?
 - Sample Keystores
 - Working with a Keystore
 - Configuring WebLogic Keystores
- Configuring WebLogic SSL



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

What Is a Keystore?

- A keystore is a repository of security certificates, either authorization certificates or public key certificates, which are used mainly in SSL encryption.
- The Java Development Kit (JDK) maintains a default CA keystore stored in
`<JAVA_HOME>/jre/lib/security/cacerts`.
 - The well-known password is “changeit”.
 - Best practice:
 - Copy the default keystore to a new location.
 - Reset the password.
 - Configure WebLogic to use the new location.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Changing the default location and password of the JDK `cacerts` keystore prevents it from being tampered with. This prevents people from adding illegitimate certificates into the keystore that could potentially cause WebLogic to trust authorities that it should not.

keytool Utility

- keytool is a standard Java SE SDK utility for managing:
 - The generation of private keys and the corresponding digital certificates
 - Keystores (databases) of private keys and the associated certificates
- The keytool utility can display certificate and keystore contents.
- You can specify an algorithm that is different from Digital Signature Algorithm (DSA) when generating digital keys by using keytool.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Oracle's keytool utility can also be used to generate a private key, a self-signed digital certificate for WebLogic, and a Certificate Signing Request (CSR). Submit the CSR to a certificate authority to obtain a digital certificate for use with WebLogic.

You can use the keytool utility to:

- Update the self-signed digital certificate with a new digital certificate
- Obtain trust and identity when using WebLogic in a production environment

Note: When you use the keytool utility, specify an algorithm that is different from the default DSA, such as RSA, because WebLogic does not support DSA.

Working with a Keystore

JDKs include a command-line tool to create, view, and modify keystore files.

- Generate a new self-signed certificate and private key and add it to a store:

```
keytool -genkeypair -alias mykey -keypass mykeypass  
-keyalg RSA -keysize 2048 -dname "CN=payroll.mycompany.com..."  
-keystore mykeys.jks -storepass mypass
```

- Import a signed certificate reply from a CA:

```
keytool -importcert -file payroll.pem -alias mykey -keypass  
mykeypass -keystore mykeys.jks -storepass mypass
```

- Inspect the contents of a store:

```
keytool -list -v -keystore mykeys.jks -storepass mypass
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The `keytool` utility is a key- and certificate-management utility. It enables users to administer their own public/private key pairs and associated certificates for use in self-authentication (where the user authenticates himself or herself to other users or services) or data integrity and authentication services by using digital signatures. It enables users to cache the public keys (in the form of certificates) of their communicating peers. `keytool` also enables users to administer secret keys used in symmetric encryption/decryption (for example, DES). `keytool` stores keys and certificates in a keystore.

Available commands include:

- `-genkeypair`: Generates a key pair (a public key and associated private key). Wraps the public key into an X.509 v3 self-signed certificate, which is stored as a single-element certificate chain. This certificate chain and the private key are stored in a new keystore entry identified by `alias`. `keyalg` specifies the algorithm used to generate the key pair, and `keysize` specifies the size of each key to be generated. `sigalg` specifies the algorithm used to sign the self-signed certificate. This algorithm must be compatible with `keyalg`. `dname` specifies the X.500 Distinguished Name to be associated with `alias`, and is used as the issuer and subject fields in the self-signed certificate. If no distinguished name is provided at the command line, the user will be prompted for one.

- `-importcert`: Reads the certificate or certificate chain (where the latter is supplied in a PKCS#7-formatted reply or a sequence of X.509 certificates) from the `cert_file` file, and stores it in the keystore entry identified by `alias`. If no file is given, the certificate or certificate chain is read from standard input. `keytool` can import X.509 v1, v2, and v3 certificates, and PKCS#7-formatted certificate chains consisting of certificates of that type. The imported data must be provided either in binary encoding format, or in printable encoding format as defined by the Internet RFC 1421 standard. In the latter case, the encoding must be bounded at the beginning by a string that starts with `-----BEGIN`, and bounded at the end by a string that starts with `-----END`. If the alias does not point to a key entry, `keytool` assumes that you are adding a trusted certificate entry. In this case, the alias should not already exist in the keystore.
If the alias does already exist, `keytool` outputs an error, because there is already a trusted certificate for that alias, and does not import the certificate. If the alias points to a key entry, `keytool` assumes that you are importing a certificate reply. When importing a certificate reply, the certificate reply is validated using trusted certificates from the keystore, and optionally using the certificates configured in the `cacerts` keystore file (if the `-trustcacerts` option was specified). If the reply is a single X.509 certificate, the `keytool` attempts to establish a trust chain, starting at the certificate reply and ending at a self-signed certificate (belonging to a root CA). The certificate reply and the hierarchy of certificates used to authenticate the certificate reply form the new certificate chain of alias. If a trust chain cannot be established, the certificate reply is not imported. In this case, the `keytool` does not print the certificate and prompt the user to verify it, because it is very hard (if not impossible) for a user to determine the authenticity of the certificate reply.
- `-list`: Prints to standard out the contents of the keystore entry identified by `alias`. If no alias is specified, the contents of the entire keystore are printed. This command by default prints the MD5 fingerprint of a certificate. If the `-v` option is specified, the certificate is printed in human-readable format, with additional information such as the owner, issuer, serial number, and any extensions. If the `-rfc` option is specified, certificate contents are printed using the printable encoding format, as defined by the Internet RFC 1421 standard.
- `-delete`: Deletes from the keystore the entry identified by `alias`. The user is prompted for the alias, if no alias is provided at the command line.

Configuring WebLogic Keystores

Settings for server1

Configuration Protocols Logging Debug Monitoring Control

General Cluster Services **Keystores** SSL Federation Service

Health Monitoring Server Start Web Services Coherence

Click the **Lock & Edit** button in the Change Center to modify the settings or

Save

Keystores: Demo Identity and Demo Trust **Change**

Identity

Demo Identity Keystore: /u01/domains/part2/wlsadmin/security/Demoidentity.jks

Demo Identity Keystore Type: jks

Demo Identity Keystore Passphrase:

Trust

Demo Trust Keystore: /u01/app/fmw/wlservr/server/lib/DemoTrust.jks

Demo Trust Keystore Type: jks

Demo Trust Keystore Passphrase:

Java Standard Trust Keystore: /u01/app/jdk1.7.0_15/jre/lib/security/cacerts

Java Standard Trust Keystore Type: jks

Java Standard Trust Keystore Passphrase:

Confirm Java Standard Trust Keystore Passphrase:

Save

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Keystores ensure the secure storage and management of private keys and trusted CAs. WebLogic is configured with a default identity keystore (`DemoIdentity.jks`) and a default trust keystore (`DemoTrust.jks`). In addition, WebLogic trusts the CA certificates in the JDK cacerts file. This default keystore configuration is appropriate for testing and development purposes. However, these keystores should not be used in a production environment.

After you configure identity and trust keystores for a WebLogic Server instance, you can configure its SSL attributes. These attributes include information about the identity and trust location for particular server instances.

Quiz

Q

The utility provided by the JDK to administer keystores is called:

- a. keystoreAdmin
- b. storekey
- c. keytool
- d. storekit

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: c

Agenda

- SSL Concepts
- WebLogic SSL Scenarios
- Keystores
- Configuring WebLogic SSL
 - Configuring SSL for WebLogic
 - SSL and WebLogic Proxy Plug-Ins
 - SSL and Hardware Load Balancers (HLB)
 - SSL and Oracle HTTP Server (OHS)



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Configuring SSL for WebLogic



The screenshot shows the Oracle WebLogic Administration Console interface. On the left, there's a tree view labeled 'Domain Structure' under 'wladmin'. The 'Servers' node is selected and highlighted with a red box. On the right, the 'Settings for server1' window is open. At the top, there are tabs: 'Configuration' (highlighted with a red box), 'Protocols', 'Logging', 'Debug', 'Monitoring', 'Control', 'Deployments', 'Services', 'Security', and 'Notes'. Below these are sub-tabs: 'General', 'Cluster', 'Services', 'Keystores', 'SSL' (highlighted with a red box), 'Federation Services', 'Deployment', 'Migration', 'Tuning', and 'Overload'. Under 'SSL', there are sections for 'Identity and Trust Locations' (with a 'Keystores' button labeled 'Change' which has a cursor over it) and 'Trust'. Both sections have 'Save' buttons at the bottom. A large padlock icon is positioned between the domain structure tree and the settings window. At the bottom of the screen, there's an 'ORACLE' logo and a copyright notice: 'Copyright © 2016, Oracle and/or its affiliates. All rights reserved.'

You configure SSL through the administration console:

1. Navigate to the server instance and click Lock & Edit.
2. Click the Configuration > SSL tabs.
3. Enter the information for the server's key that is stored in the configured keystore and click Save.

Identity and Trust Locations: Indicates where SSL should find the server's identity (certificate and private key) as well as the server's trust (trusted CAs). If set to Keystores, SSL retrieves the identity and trust from the server's key store (which is configured on the server).

Identity: Specifies the key alias and password for this server to use in the configured keystore. In this case, the demo identity alias, Demoldentity, is configured.

Trust: Specifies the keystores to use for verifying the trustworthiness of a certificate. In this case, the DemoTrust and JDK cacerts keystores are configured.

SSL and WebLogic Proxy Plug-Ins

WebLogic Server supports plug-ins for the most common web servers (Oracle iPlanet, Microsoft IIS, Apache, and Oracle HTTP Server [OHS]).

It is possible to establish secure communication between the proxy web server and WebLogic.

- Native two-way SSL is supported only via the OHS plug-in. However, plug-ins can be set up to require the client certificate and pass it on to the WebLogic Server.
 - Specific SSL directives are used to drive the behavior of requests redirected from the web server to WebLogic, for example:

```
SSLVerifyClient require  
SSLVerifyDepth 10  
SSLOptions +FakeBasicAuth +ExportCertData +CompatEnvVars
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Plug-ins are small software programs, usually written in C or C++, which enable communication and interoperability between the Java Virtual Machine and the web server native executables. Typically, WebLogic Server handles application requests that require dynamic functionality, and requests that can best be served with dynamic HTML pages, Java Server Pages (JSPs), or Servlets. The web server instead handles the requests for static pages.

In production environments, it is advisable to implement SSL encryption of the traffic going from the web server to WebLogic and vice versa.

If you are using OHS as a web server, two-way SSL encryption is supported between OHS and WebLogic.

- SSLVerifyClient require:** Configures the web server to require client certificates for connections
- SSLVerifyDepth 10:** Specifies how far the web server should follow a certificate chain while still considering the certificate as valid

- **SSLOptions** : Is a way to set other SSL options for the web server
 - **+FakeBasicAuth**: Tells the web server to translate the client certificate subject to an HTTP basic authentication username
 - **+ExportCertData**: Tells the web server to export client and server certificates to a server-side includes (SSI) or common gateway interface (CGI) environment
 - **+CompatEnvVars**: Exports additional SSI and CGI environment variables for backward compatibility

SSL and Oracle HTTP Server

The `mod_wl_ohs` module allows requests to be proxied from Oracle HTTP Server to WebLogic, whereas the `mod_weblogic` module allows requests to be proxied from Apache to WebLogic.

- `mod_wl_ohs` is installed by default with OHS and provides several benefits over `mod_weblogic`:
 - Uses Oracle's security layer (NZ library) to provide SSL support for the module. The `wlSSLWallet` directive is available to support the use of Oracle Wallets.
 - Supports two-way SSL between OHS and WebLogic Server
 - Supports IPv6 for communication with WebLogic Server
- SSL directives are specified in the `mod_wl_ohs.conf` file, which is included in the standard `httpd.conf` file.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

When using OHS as a web server proxy, there are two options for SSL configuration. In Fusion Middleware Control, from the Oracle HTTP Server menu, select *Administration > mod_wl_ohs Configuration* to access the `mod_wl_ohs` configuration page. Alternatively, you can manually modify the `mod_wl_ohs.conf` file, located in `$INSTANCE_HOME/config/OHS/<component_name>`.

The Oracle NZ library is the interface available to Oracle products for SSL protocol implementation and the cryptographic support required for it. It is a wrapper on top of Certicom and RSA BSAFE, BCERT toolkits.

SSL and Hardware Load Balancers

Hardware Load Balancers provide valuable services:

- Traffic Prioritization
- Service Failure Detection
- Caching, Optimization, and Compression
- IP Address Filtering, Content Redirection, and Content Rewriting
- SSL Offload
 - It relieves a web server of the processing burden of encrypting and/or decrypting traffic sent via SSL. The processing is offloaded to a separate device designed specifically to perform SSL encryption/decryption.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

There is a significant performance penalty associated with the use of SSL, especially in configuration that heavily relies on two-way SSL. Load balancers, which offer SSL offload, can alleviate the processing burden from the web server and/or WebLogic server by carrying out all encryption/decryption tasks.

Transactions handled over SSL can require substantial computational power to establish the connection (handshake) and then encrypt and decrypt the transferred data. SSL processing can cost five times more than clear text for the same level of transferred data. SSL offload takes much of that computing burden off the servers and places it on dedicated SSL hardware. This significantly reduces the power required and puts less strain on the servers servicing URL requests.

Quiz

Q

All WebLogic plug-ins support two-way SSL traffic between the web server and WebLogic:

- a. True
- b. False

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: b

Summary

In this lesson, you should have learned how to:

- Describe SSL concepts
- Configure keystores in WebLogic
- Configure SSL for WebLogic architectures



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Practice 8-1 Overview: Setting Up SSL

This practice covers the following topics:

- Using keytool to generate an identity keystore that contains a private key and a self-signed public certificate
- Configuring keystores by using the administration console
- Configuring SSL for a managed server



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Shared Java EE Libraries

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to configure and deploy an application as a shared Java EE library.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

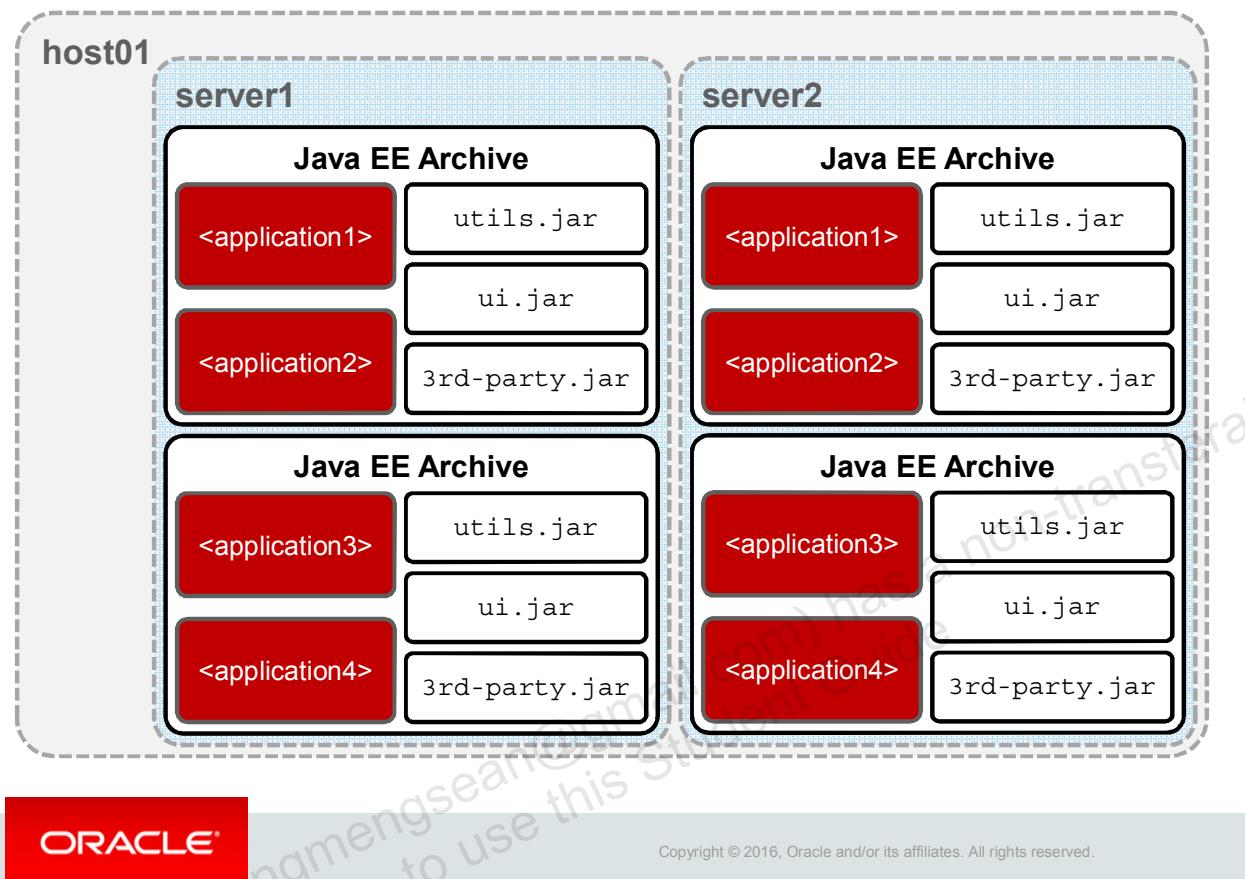
Agenda

- Shared Java EE Library Concepts
 - Before Shared Libraries
 - Shared Libraries
 - Types of Shared Libraries
 - Libraries and Deployment Configuration Precedence
 - Configuring a Shared Library
 - Referencing a Shared Library
- Deploying a Shared Library



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Before Shared Libraries



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

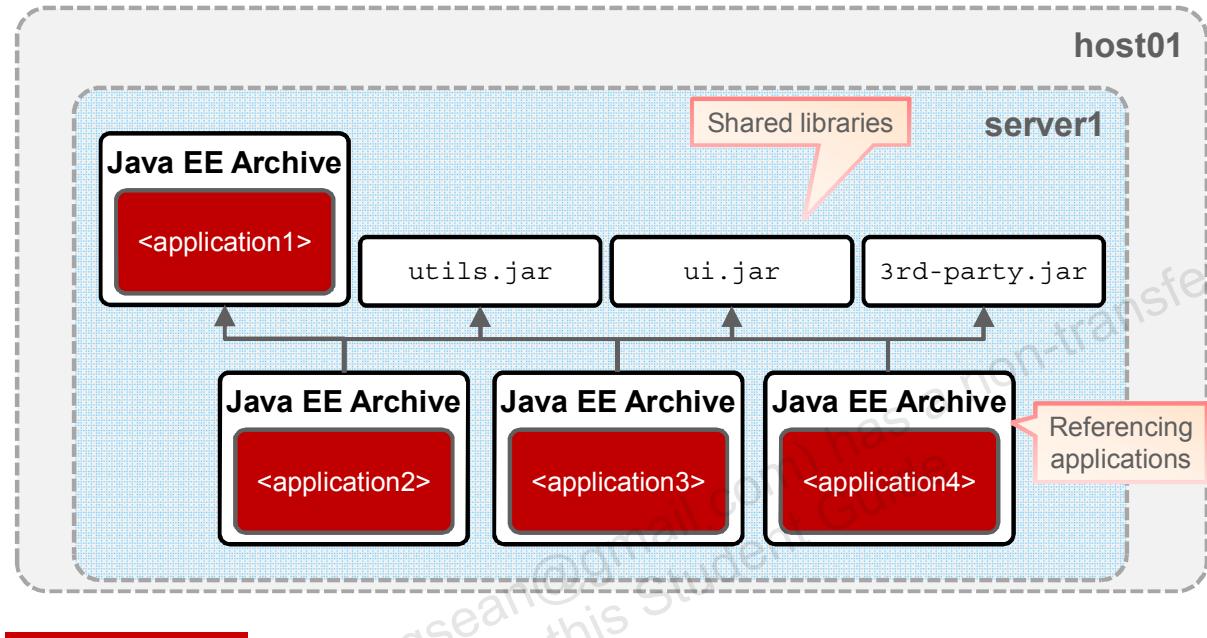


Before shared libraries existed in Java Application Servers, applications that depended on other applications and Java Archive (JAR) files containing utility or infrastructure classes required physically packaging those dependencies in the archive with the application in each deployment. This has several drawbacks including:

- Extra disk space is required to duplicate the deployment files in multiple locations.
- Extra maintenance is required to ensure that the files are included in each deployment.
- There is limited or no support for versioning of infrastructure code.
- Application code is tightly tied to infrastructure code (no separation of administration available for the different pieces that make up the overall application).

Shared Libraries

Shared libraries are collections of Java classes that are made available to applications from a single location.



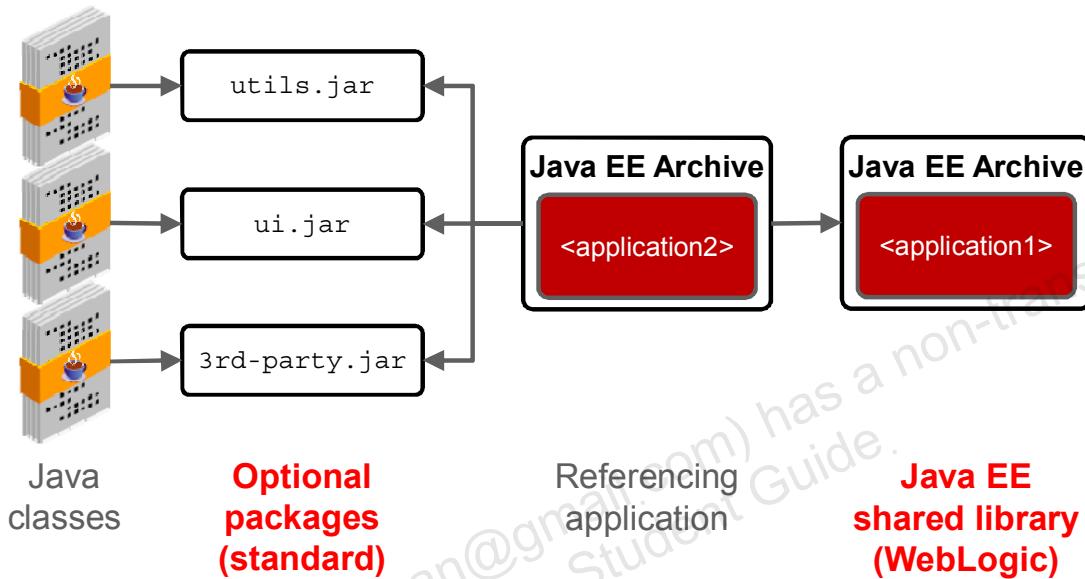
Copyright © 2016, Oracle and/or its affiliates. All rights reserved.



Shared libraries are deployed separately on the server and are designated as libraries. Applications reference the libraries they need as part of their configuration. This is realized at deployment time. A shared library's classes and configuration are merged with the referencing applications' classes and configuration during deployment.

Types of Shared Libraries

There are two types of shared libraries: optional packages and Java EE shared libraries.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

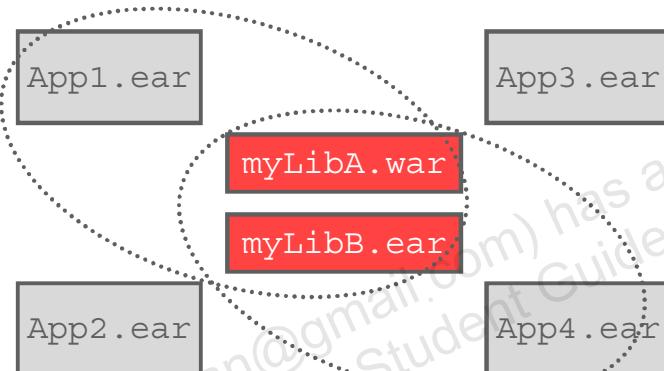
Optional packages are a Java EE standard for sharing a collection of Java classes as a JAR file for use across multiple applications. Optional packages do not support sharing Java EE components.

Java EE shared libraries are a WebLogic extension for sharing a Java EE archive for use across multiple applications.

Those are the differences between these two types of libraries. These two types of libraries are very similar in every other way, including how they are configured, deployed, and referenced. This course focuses on WebLogic's Java EE shared library features.

Java EE Library Support

- Create a library of Java EE modules, package the modules into an EAR, a WAR, or an EJB file, and then deploy and register the module with the application container.
- Other applications can later use the modules as if they were packaged in their own EAR, WAR, or EJB files.
- This allows for more reusability between the applications.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

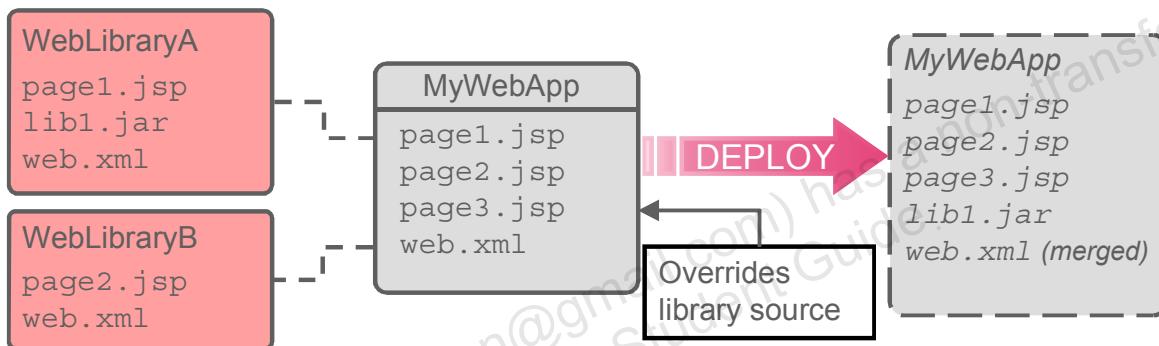
The Java EE library feature provides an easy way to share one or more types of Java EE modules among multiple enterprise applications. A Java EE library is a stand-alone EJB or web application module, multiple EJB or web application modules packaged in an Enterprise Archive (EAR), or a single plain JAR file that is registered (as a library) with the Java EE application container upon deployment.

After the library is registered, you can deploy enterprise applications that reference the library. Each referencing application receives a reference to the required library modules on deployment, and can use those modules as though they were packaged as part of the referencing application itself. The shared library classes are added to the classpath of the referencing application, and the referencing application's annotations and deployment descriptors are merged (in memory) with those of the Java EE library modules.

WebLogic Java EE Shared Libraries

A Java EE shared library:

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



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

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

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

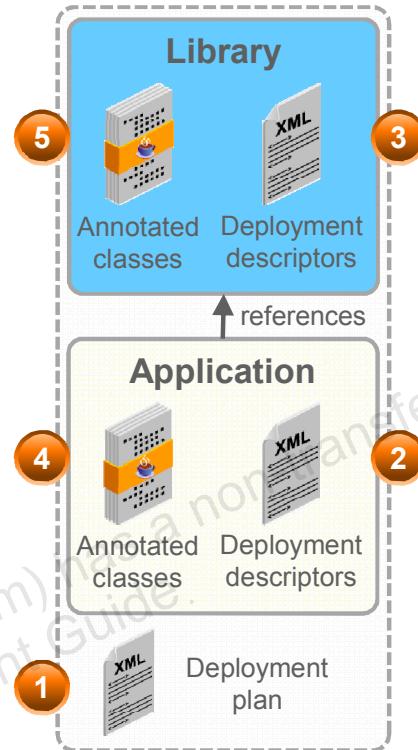
- Any file that is located in your application takes precedence over a file that is in a shared library.
- Conflicts arising between referenced libraries are resolved based on the order in which the libraries are specified in the `META-INF/weblogic-application.xml` file (for enterprise applications)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

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

Configuration Precedence and Deployment

WebLogic uses the following precedence when applying configuration properties during deployment:

1. Deployment plan
2. Application descriptors
3. Library descriptors
4. Application annotations
5. Library annotations



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

As explained in the lesson titled “Application Staging and Deployment Plans,” developers can use Java annotations when they create their applications. Some annotations are realized as configuration properties during deployment. Just as you had to consider annotations during deployment with stand-alone applications, you must also consider them when deploying applications that reference WebLogic shared Java EE libraries.

This slide shows how WebLogic processes all the configuration properties of an application that references a shared library and uses a deployment plan during deployment:

1. The deployment plan overrides any settings found in the application. This gives the administrator control over the deployment’s settings.
2. The application’s deployment descriptor overrides any properties in application annotations and in library annotations and descriptors. This lets the application control properties that are preset in the library.
3. The libraries deployment descriptor overrides any properties in the library’s annotations. Just as with application descriptors, library descriptors are a way to adapt the library to different environments.
4. The application’s annotations override library annotations.
5. The library’s annotations do not override any settings.

Managing Precedence

Remember to use the following options with appmerge to see how WebLogic is interpreting your deployments:

-writeInferredDescriptors flag

```
$ java weblogic.appmerge  
-writeInferredDescriptors  
-library jstl  
-librarydir myLibPath  
-plan myPath/plan.xml  
-output app.debug/app.ear  
MyEAR.ear
```

Command Line

metadata-complete attribute

```
<application  
  metadata-complete="true">  
  ...  
</application>
```

application.xml

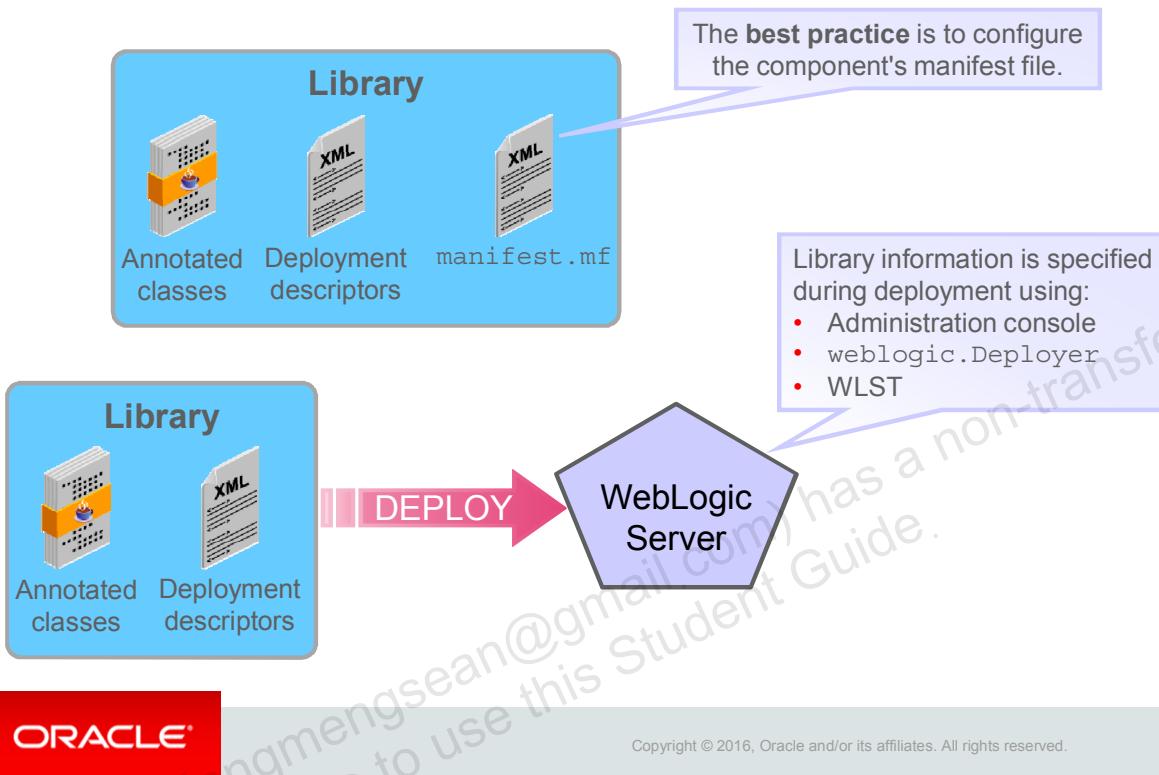


Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic's appmerge tool enables you to see how the container views the configuration of your deployments that include shared libraries. As an administrator, you will find using appmerge to be a productive tool for seeing how WebLogic processes the configuration properties of your deployments. Note that now you can include the arguments for shared libraries and a deployment plan.

Configuring a Shared Library

There are two ways to configure a shared library:



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

You indicate to WebLogic when a deployment is a library and the configuration of that library. The configuration is done in one of two ways:

- The deployment's manifest file is the best way to configure a library because it stays with the deployment and eliminates any confusion about which version of the library is represented by the archive. This always guarantees that a library is deployed with version information.
- The various deployment tools available in WebLogic enable you to configure library settings. These settings are realized during deployment and occur at deployment by entering the information in the tool that performs the deployment. This information is not stored with the application files at all and there is no guarantee that the library is deployed with version information.

Configuring a Shared Library in the Manifest File

Configuring a manifest file:

- jstl-1.2.war

```
Manifest-Version: 1.0
...
Specification-Version: 1.2
Implementation-Title: JSTL Reference Implementation
Implementation-Version: 1.2.0.1
Implementation-Vendor: Sun Microsystems, Inc.
Extension-Name: jstl
```

Shared library name and version

MANIFEST.MF

- jsf-2.0.war

```
Manifest-Version: 1.0
...
Specification-Version: 2.0
Implementation-Title: JSF Reference Implementation
Implementation-Version: 1.0.0.0_2-0-2
Original-Implementation-Version-From-RI: it will use glassfish jsf 2.1
.5 from the system classpath
Implementation-Vendor: Sun Microsystems, Inc.
Extension-Name: jsf
```

MANIFEST.MF



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

When you configure a shared library, you specify a name, a specification version, and an implementation version. This is true regardless of the method used to configure the library. WebLogic supports versioning of shared libraries, so that referencing applications can specify a required minimum version of the library to use or an exact, required version. WebLogic supports two levels of versioning for shared libraries:

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

As a best practice, you should always include version information (an implementation version, or both an implementation and a specification version) when creating shared libraries. Creating and updating version information as you develop shared components enable you to deploy multiple versions of those components simultaneously for testing and to support applications that use each version.

Referencing a Shared Library

- For web applications, list the required shared libraries in `weblogic.xml`.
- For enterprise applications, list the required shared libraries in `weblogic-application.xml`.
- Excerpts from `weblogic.xml`:

```
<library-ref>
    <library-name>jstl</library-name>
    <specification-version>1.2</specification-version>
    <implementation-version>1.2.0.1</implementation-version>
    <exact-match>false</exact-match>
</library-ref>

<library-ref>
    <library-name>jsf</library-name>
    <specification-version>2.0</specification-version>
    <implementation-version>1.0.0.0_2-0-2</implementation-version>
    <exact-match>true</exact-match>
</library-ref>
```

Shared library
name and version



weblogic-application.xml



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

You configure an application reference to a library within its WebLogic-based deployment descriptor by using the `<library-ref>` XML tag. The example in this slide shows an enterprise application configured to reference both the `jstl` and `jsf` libraries shown in the previous slide. WebLogic processes library references depending upon how the version information contained in the library and the referencing application are configured:

- `specification-version`:
 - If not set, the referencing application uses a matching library with the highest `specification-version`.
 - If a value is set that uses the major and minor version format, the application uses a matching library with the highest `specification-version` that is at least as high as this setting. If all available libraries are below this setting, the application cannot deploy.
 - If a value is set that does not adhere to the major and minor version format, such as `1.2.beta`, the application requires an exact match of the `specification-version` setting.

- `implementation-version`:
 - If not set, the referencing application uses a matching library with the highest `implementation-version`.
 - If a value is set that uses the major and minor version format, the application uses a matching library with the highest `implementation-version` that is at least as high as this setting. If all available libraries are below this setting, the application cannot deploy.
 - If a value is set that does not adhere to the major and minor version format, such as `1.2.beta`, the application requires an exact match of the `implementation-version` setting.
- `exact-match`:
 - When set to the default value of `false`, the application will use a matching library with the highest version available.
 - When set to `true`, the application requires a matching library with the exact same `specification-version` and `implementation-version`.

Quiz



Which of the following is a correct sequence of precedence for deployment properties when an application that references a shared library is deployed?

- a. Application, then library, then deployment plan properties
- b. Library, then application, then deployment plan properties
- c. Deployment plan, then library, then application properties
- d. Deployment plan, then application, then library properties

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: d

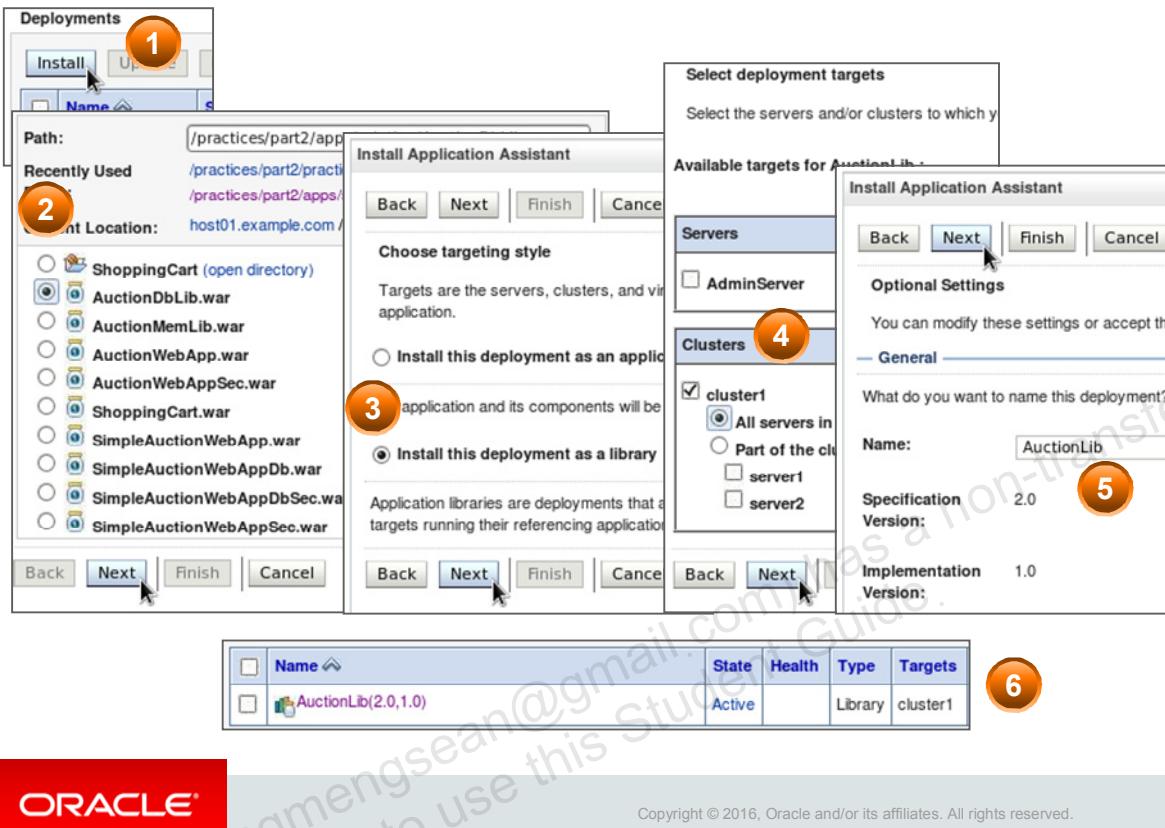
Agenda

- Shared Java EE Library Concepts
- Deploying a Shared Library
 - With the Administration Console
 - With `weblogic.Deployer`
 - With WLST



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Deploying a Library with the Console



Deploying a shared library using the administration console is like deploying an application. The only difference is that in the “Choose targeting style section,” you select “Install this deployment as a library” to tell WebLogic how to deploy the library. All the library name, specification-version, and implementation-version information during deployment is derived from the configuration in the deployment’s manifest file. You deploy a library by performing the following steps:

1. Click install in the administration console to begin the deployment process.
2. Select the application to deploy as a library and click Next.
3. Select “Install this deployment as a library” and click Next.
4. Select the targets where the library is deployed and click Next.
5. Review the deployment settings, verify that the library configuration is correct, and click Next or Finish.
6. Verify that your library is deployed and active, and that it has version information associated with it in the name column.

Deploying a Library with `weblogic.Deployer`

You can deploy shared libraries by using `weblogic.Deployer` library options:

- Without version information:

```
$ java weblogic.Deployer  
-adminurl http://localhost:7001 -username weblogic  
-password Welcome1 -deploy -targets cluster1  
-library /deployments/AuctionDbLib.war
```

Command Line

- With version information:

```
$ java weblogic.Deployer  
-adminurl http://localhost:7001 -username weblogic  
-password Welcome1 -deploy -targets cluster1  
-library -libspecver 2.0  
-libimplversion 1.0 /deployments/AuctionDbLib.war
```

Command Line



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The `weblogic.Deployer` tool will deploy your libraries whether the version configuration is contained within the manifest or not:

- If the manifest does not contain version information, the library is deployed with no version information. If this occurs, the library must be undeployed before deploying it with version information.
- If the manifest does contain version information, the library is deployed with that configuration.
- If version information is included on the command line when executing `weblogic.Deployer` and the version information is different than what is contained in the manifest file, the deployment fails.

Deploying a Library with WLST

You can deploy shared libraries by using WLST library options:

- Without version information:

```
> deploy('AuctionLib',
          path=/deployments/AuctionDbLib.war,
          targets='cluster1', libraryModule='true')
```

WLST Prompt

- With version information:

```
> deploy('AuctionLib',
          path=/deployments/AuctionDbLib.war,
          libSpecVersion='2.0', libImplVersion='1.0',
          targets='cluster1', libraryModule='true')
```

WLST Prompt



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WLST works the same as `weblogic.Deployer`.

Summary

In this lesson, you should have learned how to configure and deploy an application as a shared Java EE library.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Practice 9-1 Overview: Configuring and Deploying a Shared Library

This practice covers the following topics:

- Configuring two shared libraries by using their manifest files:
 - AuctionLib: spec: 1.0, impl: 1.0
 - AuctionMemLib: Data stored in memory
 - AuctionLib: spec: 2.0, impl: 1.0
 - AuctionDbLib: Data stored in database
- Deploying shared libraries by using the administration console and weblogic.Deployer
- Deploying an application that references a shared library
- Testing the application



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Sean Tay (taysiangmengsean@gmail.com) has a non-transferable
license to use this Student Guide.

Application Work Managers

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe how WebLogic handles concurrent client requests
- Monitor thread pool size and usage
- Prioritize application processing by using work managers
- Tune work managers by using request classes and constraints
- Update an application to use a work manager



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Agenda

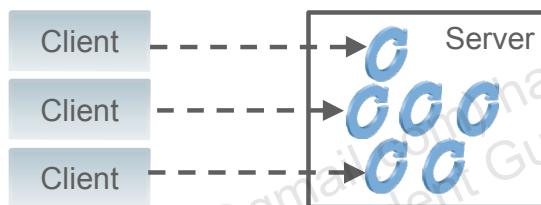
- Default WebLogic Request Handling
- Work Manager Concepts
- Creating a Work Manager



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic Server Threads

- Like all server software, WebLogic processes concurrent requests by assigning each to a thread from an available pool.
- WebLogic is “self-tuning” and automatically adjusts the number of threads in the pool as needed.
- Some threads are dedicated to various background tasks.
- By default, all requests are given equal priority.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

WebLogic uses a single thread pool in which all types of work are executed. WebLogic prioritizes work based on rules that you define, and runtime metrics, including the actual time it takes to execute a request and the rate at which requests enter and leave the pool.

The common thread pool changes its size automatically to maximize throughput. The incoming request queue monitors throughput over time and, based on history, determines whether to adjust the thread count. For example, if historical throughput statistics indicate that a higher thread count increased throughput, WebLogic increases the thread count. Similarly, if statistics indicate that fewer threads did not reduce throughput, WebLogic decreases the thread count. This new strategy makes it easier for administrators to allocate processing resources and manage performance, avoiding the effort and complexity involved in configuring, monitoring, and tuning custom executes queues.

Monitoring a Server Thread Pool

Print the current stack trace for each thread.

View current thread pool statistics.

Active Execute Threads	Execute Thread Total Count	Execute Thread Idle Count	Queue Length	Pending User Request Count	Completed Request Count
1	5	0	0	0	735

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Select a server and click Monitoring > Threads. The first table provides general information about the status of the thread pool. The second table provides information about individual threads. The available columns in the first table include:

- **Execute Thread Total Count:** The total number of threads in the pool
- **Execute Thread Idle Count:** The number of idle threads in the pool. This count does not include standby threads and stuck threads. The count indicates threads that are ready to pick up new work when it arrives.
- **Pending User Request Count:** The number of pending user requests in the priority queue. The priority queue contains requests from internal subsystems and users. This is just the count of all user requests.
- **Hogging Thread Count:** Returns the threads that are being hogged by a request right now. These threads will either be declared as stuck after the configured timeout or will return to the pool before that. The self-tuning mechanism will backfill if necessary.
- **Throughput:** The mean number of requests completed per second

To display the current Java stack trace for active threads, click the Dump Thread Stacks button.

Monitoring Server Threads

C
C

Name ↗	Total Requests	Transaction	User	Idle	Stuck
[ACTIVE] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'	1199		<WLS Kernel>	false	false
[STANDBY] ExecuteThread: '1' for queue: 'weblogic.kernel.Default (self-tuning)'	234		<WLS Kernel>	true	false

Current status and statistics for each thread

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The second table on the server's Monitoring > Threads tab provides the status and statistics for individual threads, including:

- **Total Requests:** The number of requests that have been processed by the thread
- **Current Request:** A String representation of the request this thread is currently processing
- **Transaction:** The XA transaction on whose behalf the execute thread is currently working
- **User:** The name associated with this thread
- **Idle:** Returns the value "true" if the execute thread has no work assigned to it
- **Stuck:** Returns "true" if the execute thread is being hogged by a request for much more than the normal execution time as observed by the scheduler automatically. If this thread is still busy after the stuck thread max time, it is declared as stuck.

Stuck Thread Handling

- Threads that are running for a long time are likely to be deadlocked or in an infinite loop, or indicate an overloaded server.
- WebLogic Server:
 - Periodically checks how long threads have been running
 - Logs a warning if a thread becomes stuck
 - Creates additional threads to handle new requests
 - Sets the server's state to Failed after a specified number of threads become stuck
 - Can automatically shut itself down if in the Failed state



ORACLE®

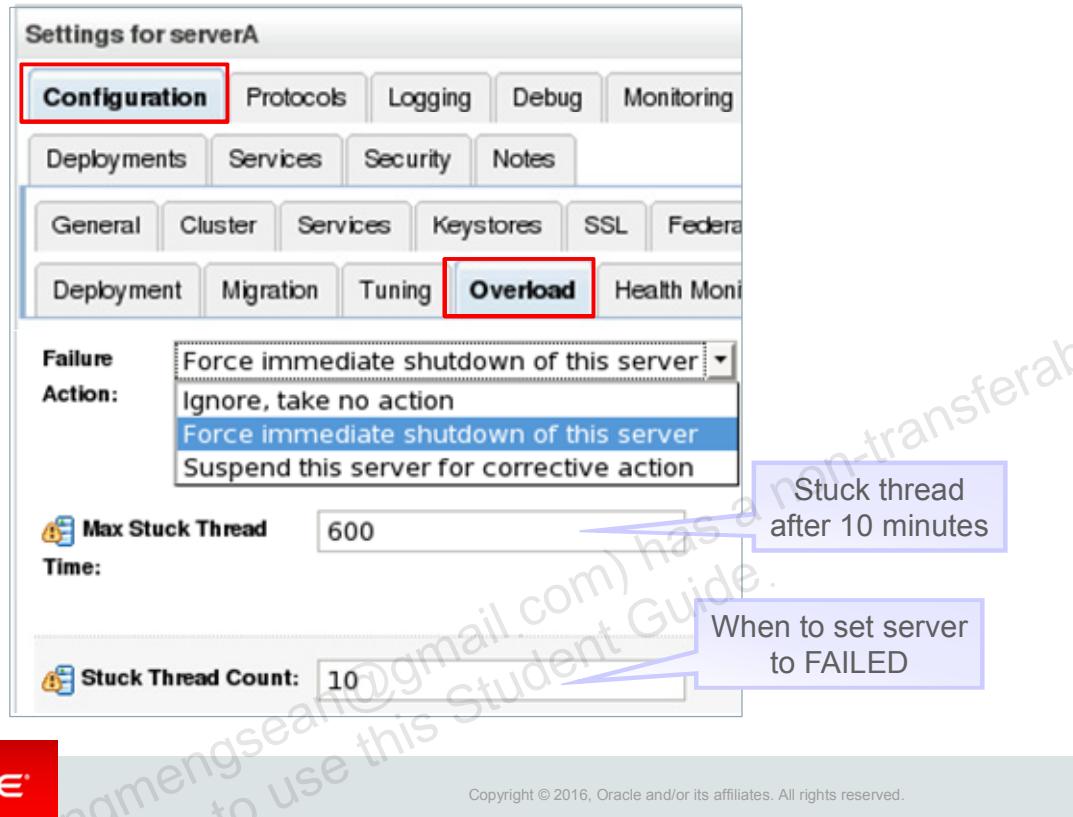
Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

If an application server continues to accept requests after system capacity is reached, application performance and stability can deteriorate. WebLogic Server diagnoses a thread as stuck if it is continually working (not idle) for a set period of time. You can tune a server's thread detection behavior by changing the length of time before a thread is diagnosed as stuck, and by changing the frequency with which the server checks for stuck threads.

If all application threads are stuck, a server instance marks itself failed and, if configured to do so, exits. You can configure Node Manager, or a third-party, high-availability solution to restart the server instance for automatic failure recovery. You can configure these actions to occur when not all threads are stuck, but when the number of stuck threads have exceeded a configured threshold:

- Shut down the work manager if it has stuck threads. A work manager that is shut down will refuse new work and reject existing work in the queue by sending a rejection message. In a cluster, clustered clients will fail over to another cluster member.
- Shut down the application if there are stuck threads in the application. The application is shut down by bringing it into administration mode. All work managers belonging to the application are shut down, and behave as described above.

Configuring Stuck Thread Handling



Select a server and locate its Configuration > Overload tab. The following fields relate to stuck thread handling:

- **Shared Capacity For Work Managers:** The total number of requests that can be present in the server. This includes requests that are in the queue and awaiting execution as well as those under execution. The server performs a differentiated denial of service on reaching the shared capacity. A request with higher priority will be accepted in place of a lower priority request already in the queue. The lower priority request is kept waiting in the queue until all high priority requests are executed. Additional lower priority requests are rejected right away.
- **Failure Action:** Enable automatic shutdown or suspension of the server on a failed state. The server self-health monitoring detects fatal failures and marks the server as failed.
- **Max Stuck Thread Time:** The number of seconds that a thread must be continually working before this server considers the thread stuck
- **Stuck Thread Count:** The number of stuck threads after which the server is transitioned into FAILED state. Mark the server instance as failed and shut it down if there are stuck threads in the server. In a cluster, clustered clients that are connected or attempting to connect will fail over to another cluster member.

Application Stuck Thread Handling

Individual applications can automatically transition into administration mode when stuck threads are detected.

```
<application-admin-mode-trigger>
  <max-stuck-thread-time>300</max-stuck-thread-time>
  <stuck-thread-count>5</stuck-thread-count>
</application-admin-mode-trigger>
```

weblogic-application.xml



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The `<application-admin-mode-trigger>` element of `weblogic-application.xml` specifies the number of stuck threads needed to bring the application into administration mode. Similar to server-level stuck thread detection, you must specify the amount of time, in seconds, for which a thread must execute an application request before being considered stuck.

Quiz



For which of the following levels in WebLogic is stuck thread handling available?

- a. Domain and application scope
- b. Domain and server scope
- c. Server and machine scope
- d. Server and application scope

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: d

Agenda

- Default WebLogic Request Handling
- Work Manager Concepts
 - Work Managers
 - Work Manager Scope
 - Work Manager Architecture
- Creating a Work Manager



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Work Managers

- Service-level agreements (SLAs) describe performance requirements for a specific application or component:
 - Percentage of server resources to use
 - Maximum response time
 - Maximum number of threads to use
- An SLA is implemented on WebLogic by using a work manager.
- Applications use the default work manager if they are not assigned to a specific work manager.
- To override the SLA characteristics of the default work manager, create one named “default”.



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

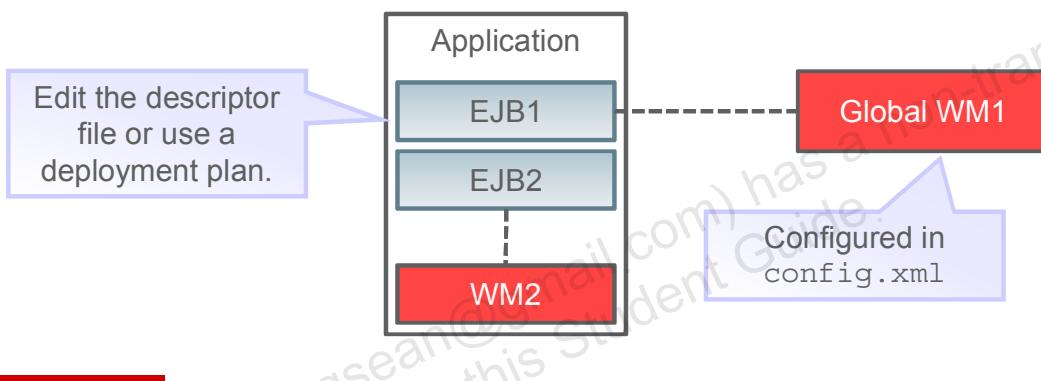
WebLogic allows you to configure how your application prioritizes the execution of its work. Based on rules that you define and by monitoring actual runtime performance, WebLogic can optimize the performance of your application and maintain SLAs. You tune the thread utilization of a server instance by defining rules and constraints for your application by defining a work manager and applying it either globally to WebLogic domain or to a specific application component. Each distinct SLA requirement needs a unique work manager.

To handle thread management and perform self-tuning, WebLogic implements a default work manager. This work manager is used by an application when no other work managers are specified in the application's deployment descriptors. In many situations, the default work manager may be sufficient for most application requirements. WebLogic's thread-handling algorithms assign to each application its own fair share by default. Applications are given equal priority for threads and are prevented from monopolizing them.

You can override the behavior of the default work manager by creating and configuring a global work manager called “default.” This enables you to control the default thread-handling behavior of WebLogic.

Work Manager Scope

- Work managers are associated with applications by using XML deployment descriptors.
- Global work managers are defined in the console and are available to any application.
- Application-scoped work managers are defined “inline” within the same deployment descriptors.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

You can create global work managers that are available to all applications and modules deployed on a server in the WebLogic administration console and in `config.xml`. An application uses a globally defined work manager as a template. Each application creates its own instance that handles the work associated with that application and separates that work from other applications. This separation is used to handle traffic directed to two applications that are using the same dispatch policy. Handling each application's work separately allows an application to be shut down without affecting the thread management of another application. Although each application implements its own work manager instance, the underlying components are shared.

In addition to globally scoped work managers, you can also create work managers that are available only to a specific application or module. You can define application-scoped work managers in the WebLogic administration console and in the following descriptors: `weblogic-application.xml`, `weblogic-ejb-jar.xml`, and `weblogic.xml`. If you do not explicitly assign a work manager to an application, it uses the default work manager.

Work Manager Architecture

- Request classes describe the guidelines that WebLogic uses to allocate threads from the pool.
- Constraints define additional boundary conditions that limit when threads are allocated (can override request class).
- Error-handling policies define how to handle deadlocked or timed-out threads.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

A request class expresses a scheduling guideline that WebLogic uses to allocate threads to requests. Request classes help ensure that high-priority work is scheduled before less important work, even if the high-priority work is submitted after the low-priority work. WebLogic takes into account how long it takes for requests to each module to complete.

A constraint defines the minimum and maximum number of threads allocated to execute requests and the total number of requests that can be queued or executed before WebLogic begins rejecting requests.

In response to stuck threads, you can define an error-handling policy that shuts down the work manager, moves the application into administration mode, or marks the entire server instance as failed.

Quiz



In which of the following files is the configuration for globally-scoped work managers located?

- a. config.xml**
- b. weblogic.xml**
- c. config/workmanager-<num>.xml**
- d. weblogic-application.xml**

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: a

Agenda

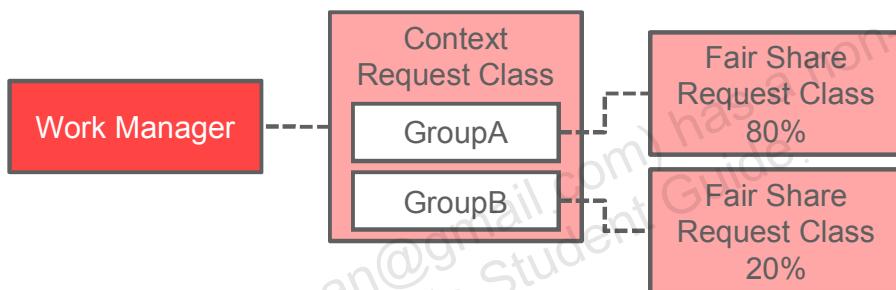
- Default WebLogic Request Handling
- Work Manager Concepts
- Creating a Work Manager
 - Request Classes
 - Creating a Work Manager
 - Creating a Request Class
 - Constraints
 - Creating a Constraint
 - Work Manager WLST Example
 - Work Managers and Stuck Threads
 - Assigning Work Managers to Applications



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Request Classes

Class Type	Description
Fair Share	A numeric weight that represents the average percentage of thread execution time while under load (default=50)
Response Time	A target average response time, in milliseconds
Context	Associate other request classes with specific security principals (users, groups)



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

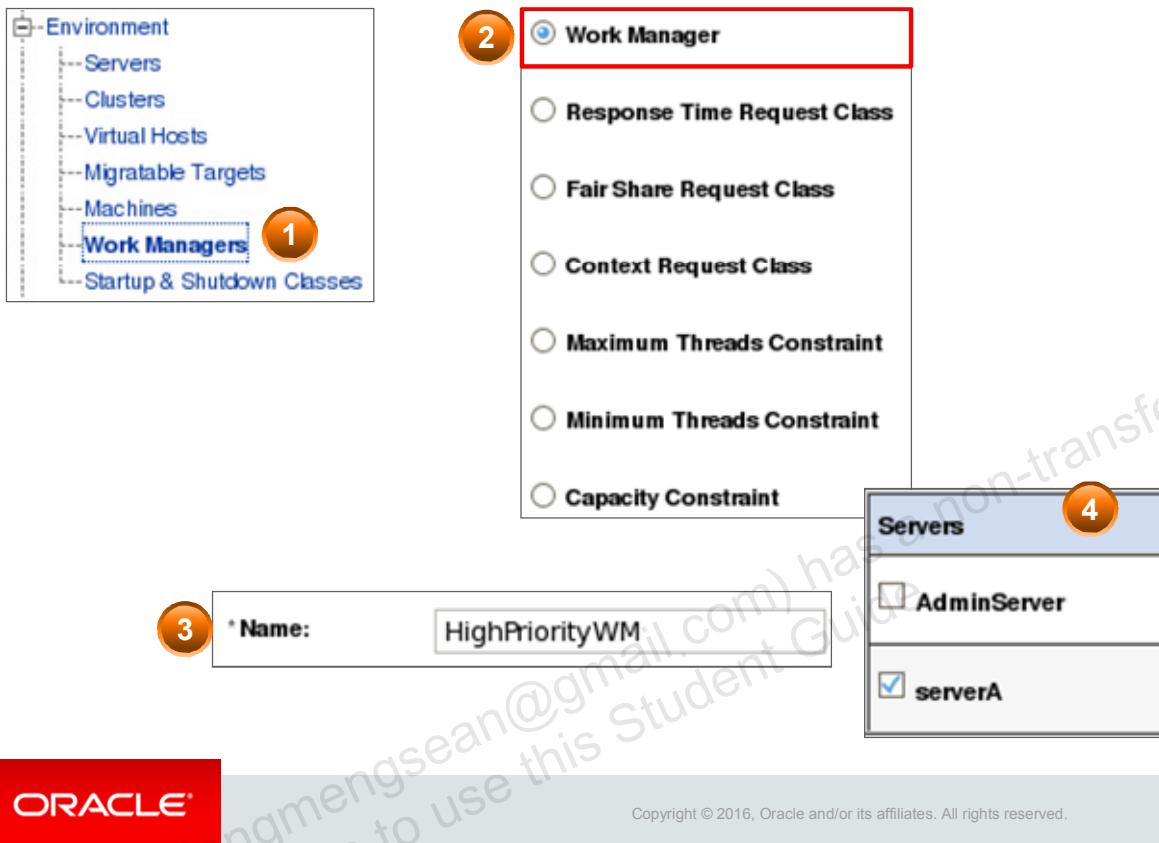
There are multiple types of request classes, each of which expresses a scheduling guideline in different terms. A work manager may specify only one request class.

A fair share request class specifies the average thread-use time required to process requests. For example, assume that WebLogic is running two modules. The work manager for Module1 specifies a fair share of “80” and the work manager for Module2 specifies a fair share of “20.” During a period of sufficient demand, with a steady stream of requests for each module such that the number requests exceed the number of threads, WebLogic will allocate 80% and 20% of the thread-usage time to Module1 and Module2, respectively.

A response time request class specifies a response time goal in milliseconds. Response time goals are not applied to individual requests. Instead, WebLogic computes a tolerable waiting time for requests with that class by subtracting the observed average thread use time from the response time goal, and schedules requests so that the average wait for requests with the class is proportional to its tolerable waiting time.

A context request class aggregates one or more other request classes, and assigns each to a unique username or group.

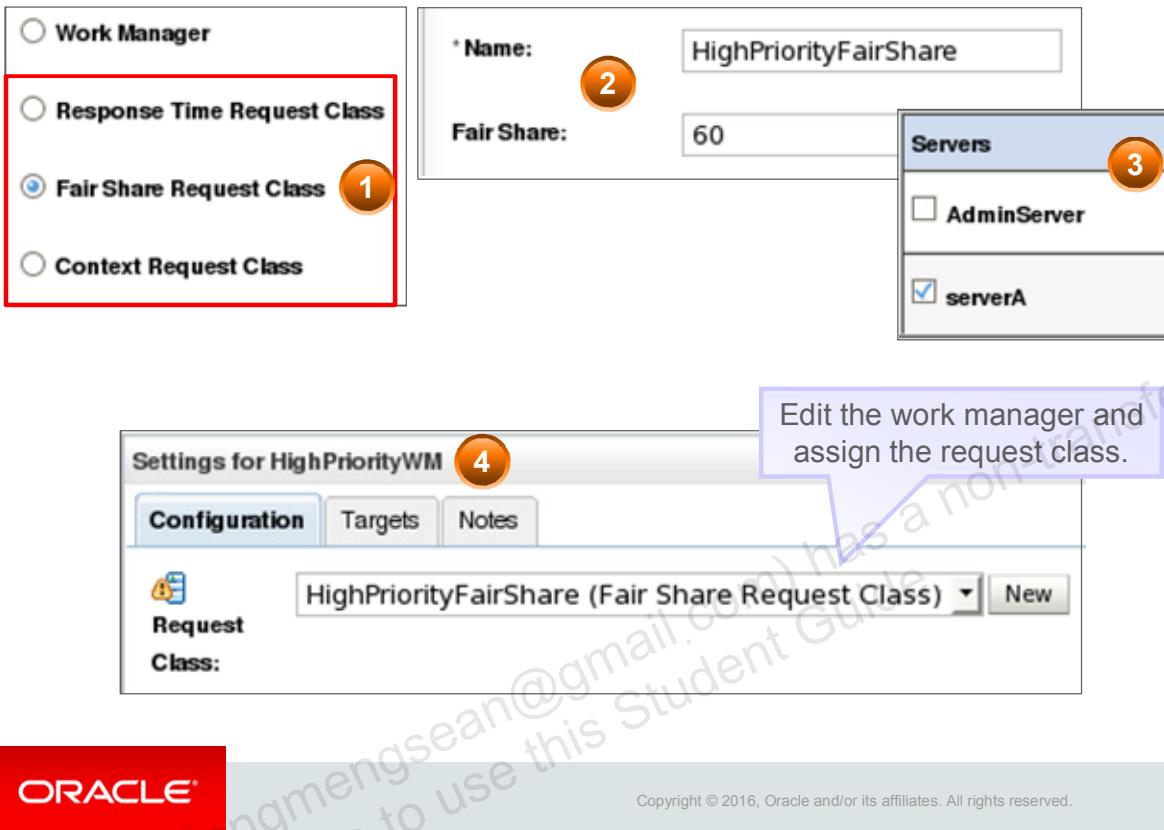
Creating a Work Manager



Using the administration console, you can create global work managers that are used to prioritize thread execution. To create a global work manager:

1. In the left pane of the console, expand Environment and select Work Managers. Click New.
2. Select Work Manager, and click Next.
3. In the Name field, enter a name for the new work manager. Click Next.
4. In the Available Targets list, select the server instances or clusters on which you will deploy applications that reference the work manager. Then click Finish.

Creating a Request Class



After you have created a global work manager, you typically create at least one request class or constraint and assign it to the work manager. Each work manager can contain only one request class, but you can share request classes among multiple work managers. To create a global request class:

1. In the left pane of the console, expand Environment and select Work Managers. Click New and then select the type of global request class that you want to create. Click Next.
2. For a fair share request class, enter the numeric weight in the Fair Share field. For a response time request class, enter a time in milliseconds in the Response Time Goal field. When finished, click Next.
3. In the Available Targets list, select the server instances or clusters on which you will deploy applications that reference this request class. Then click Finish.
4. Edit your existing work manager. Select your new request class by using the Request Class field, and click Save.

Constraints

Constraint Type	Description
Maximum Threads	<ul style="list-style-type: none"> The maximum number of concurrent threads that can be used to service requests Can be a static value or can dynamically adjust to the maximum capacity of a given JDBC data source
Minimum Threads	The minimum number of threads that the server should allocate
Capacity	The maximum size of the request backlog before rejecting new requests



ORACLE®

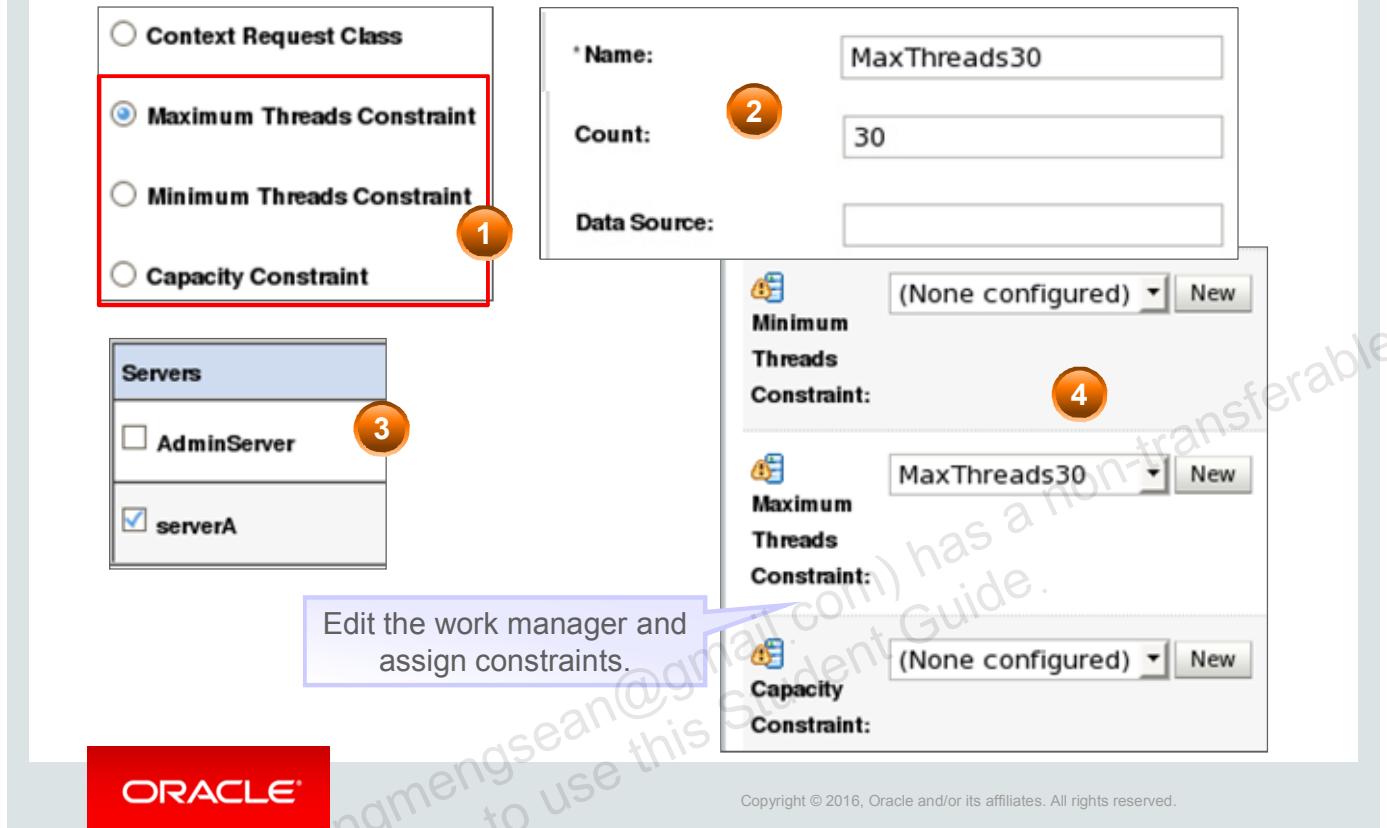
Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

A maximum threads constraint limits the number of concurrent threads executing requests from the constrained work set. The default is unlimited. For example, consider a constraint defined with maximum threads of 10 and shared by three entry points. The scheduling logic ensures that not more than 10 threads are executing requests from the three entry points combined. You can define a static value for this constraint or in terms of the availability of a dependent resource, such as a JDBC data source connection pool.

A minimum threads constraint guarantees the number of threads the server will allocate to affected requests to avoid deadlocks. The default is zero. This constraint might not necessarily increase a fair share. This type of constraint has an effect primarily when the server instance is close to a deadlock condition. In that case, the constraint will cause WebLogic to schedule a request even if requests in the service class have gotten more than its fair share recently.

A capacity constraint causes the server to reject requests only when it has reached its capacity. The default is -1. Note that the capacity includes all requests, queued or executing, from the constrained work set. Work is rejected either when an individual capacity threshold is exceeded or if the global capacity is exceeded. This constraint is independent of the global queue threshold.

Creating a Constraint



To create a global constraint and assign it to a work manager:

1. In the left pane of the console, expand Environment and select Work Managers. Click New and then select the type of global constraint that you want to create. Click Next.
2. Enter the configuration information based on the type of constraint you are creating. For a maximum or minimum threads constraint, enter a thread count. Alternatively, for a maximum threads constraint, use the Data Source field to supply the name of a JDBC data source. The maximum capacity of the data source is then used as the constraint.
3. In the Available Targets list, select server instances or clusters on which you will deploy applications that reference this constraint.
4. After you have created a constraint, you must assign it to an existing work manager to use it. Each work manager can contain only one constraint of each type, but you can share constraints among multiple work managers.

Work Manager WLST Example

Create a global work manager and a request class:

```
edit()
startEdit()

targetServer = getMBean('/Servers/serverA')
tuning = getMBean('/SelfTuning/MyDomain')
wm = tuning.createWorkManager('HighPriorityWM')
wm.addTarget(targetServer)
rq = tuning.createFairShareRequestClass('HighPriorityFairShare')
rq.setFairShare(60)
rq.addTarget(targetServer)
wm.setFairShareRequestClass(rq)

save()
activate(block='true')
```

WLST Script



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Refer to the following MBeans for more information about MBeans associated with work managers:

- SelfTuningMBean
- WorkManagerMBean
- WorkManagerShutdownTriggerMBean
- FairShareRequestClassMBean
- ResponseTimeRequestClassMBean
- ContextRequestClassMBean
- MaxThreadsConstraintMBean
- MinThreadsConstraintMBean
- CapacityMBean

Work Managers and Stuck Threads

- WLS can automatically shut down a work manager when it detects that the work manager has stuck threads.
- For global work managers, you must use WLST.

Configure stuck thread detection for a global work manager:

```
...
wmShutdown = getMBean('/SelfTuning/MyDomain/WorkManagers/wm1/
    WorkManagerShutdownTrigger/wm1')
wmShutdown.setStuckThreadCount(5)
wmShutdown.setMaxStuckThreadTime(300)
```

WLST Script



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The `WorkManagerShutdownTriggerMBean` MBean configures the conditions under which an associated work manager is automatically shut down. The trigger specifies the number of threads that need to be stuck for a certain amount of time. A shutdown work manager refuses new work but attempts to complete pending work. There is currently no interface in the administration console for configuring this MBean type for global work managers. For application-scoped work managers, use the `<work-manager-shutdown-trigger>` element, which is a child element of `<work-manager>`.

Assigning Work Managers to Applications

Assign a work manager to a web application or module by using `weblogic.xml`:

```
...  
<wl-dispatch-policy>HighPriorityWM</wl-dispatch-policy>
```

`weblogic.xml`

Assign a work manager to a specific EJB by using `weblogic-ejb-jar.xml`:

```
...  
<weblogic-enterprise-bean>  
  <ejb-name>AccountManager</ejb-name>  
  ...  
  <dispatch-policy>HighPriorityWM</dispatch-policy>  
</weblogic-enterprise-bean>
```

`weblogic-ejb-jar.xml`



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

In your deployment descriptors, you reference one of the work managers, request classes, or constraints by its name.

An enterprise application (EAR) cannot be directly associated with a work manager, although it can define its own application-scoped work managers. Instead, individual modules within the enterprise application can reference global or application-scoped work managers.

For web or web-service applications, use the `wl-dispatch-policy` element to assign the web application to a configured work manager by identifying the work manager name. This web application-level parameter can be overridden at the individual servlet or JSP level by using the `per-servlet-dispatch-policy` element.

For EJB applications, use the `dispatch-policy` element to assign individual EJB components to specific work managers. If no `dispatch-policy` is specified, or the specified `dispatch-policy` refers to a nonexistent work manager, the server's default work manager is used instead.

Quiz



Which of the following is NOT a type of work manager request class?

- a. Context
- b. Fair Share
- c. Response Time
- d. Cluster

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Answer: d

Summary

In this lesson, you should have learned to:

- Describe how WebLogic handles concurrent client requests
- Monitor thread pool size and usage
- Prioritize application processing by using work managers
- Tune work managers by using request classes and constraints
- Update an application to use a work manager



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Practice 10-1 Overview: Creating and Using Work Managers

This practice covers the following topics:

- Configuring server work managers
- Assigning request classes to work managers
- Using deployment plans to associate work managers with applications
- Verifying work manager parameters during a stress test



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Sean Tay (taysiangmengsean@gmail.com) has a non-transferable
license to use this Student Guide.