Oracle WebLogic Server 11*g*: Advanced Administration

Volume II • Student Guide

D58686GC20
Edition 2.0
December 2010
D71659

ORACLE

Author

TJ Palazzolo

Technical Contributors and Reviewers

Prashant Agarwal
Tom Barnes
Steve Button
Dave Cabelus
Josh Dorr
Arvind Jain
Anthony Lai
Serge Moiseev
Craig Perez
Mark Prichard
Sandeep Shrivastava
John Van Pelt

Editor

Malavika Jinka

Graphic Designer

Maheshwari Krishnamurthy

Publishers

Revathi Ramamoorthy
Srividya Rameshkumar

Contents

1 Course Overview
Objectives 1-2
Target Audience 1-3
Introductions 1-4
Course Schedule 1-5
Course Practices 1-7
Classroom Guidelines 1-8
For More Information 1-9
Oracle by Example (OBE) 1-10
Related Training 1-11

2 Installation Management
Objectives 2-2
WebLogic Server Review 2-3
WebLogic Smart Update 2-4
WebLogic Patch Terminology 2-5
Getting Started 2-6
Downloading and Applying Patches 2-7
WebLogic Start Script Review 2-8
Patch Classpath 2-9
Start Script Editor 2-10
Smart Update Command-Line Interface 2-11
Patch Distribution 2-12
Oracle Cloning Tool 2-13
Quiz 2-14
Summary 2-16

3 Domain Templates
Objectives 3-2
Road Map 3-3
Domain Template Review 3-4
Template Contents 3-5
Template Exclusions 3-6
Script Replacement Variables 3-7
Template SQL Scripts 3-8

Extension Template Concepts 3-10
Fusion Middleware (FMW) Templates 3-11
Section Summary 3-12
Road Map 3-13
Creating a Domain Template 3-14
Selecting the Template Domain Source 3-15
Selecting the Template Destination: Adding Files to the Template 3-16
Adding SQL Scripts to the Template 3-17
Configuring the Administration Server 3-18
Defining Users, Groups, and Roles 3-19
Configuring Group Membership 3-20
Configuring Role Membership 3-21
Creating Windows Start Menu Entries 3-22
Preparing Scripts and Files with Variables 3-23
Creating an Extension Template 3-24
Section Summary 3-26
Quiz 3-27
Lesson Summary 3-29
Practice 3-1: Create a Custom Domain Template 3-30

4 Other Domain Tools
Objectives 4-2
Review of Basic WLST Commands 4-3
WLST Best Practice: Variable Declaration 4-4
WLST Best Practice: Password Management 4-5
WLST Best Practice: Error Handling 4-6
Working with Templates Using WLST 4-7
Template WLST Examples 4-8
Pack Tool 4-9
Template Types 4-10
Creating Managed Server Templates 4-11
Unpack Tool 4-13
ConfigToScript 4-14
Generated Script Internals 4-15
Domain Configuration Archives 4-16
WebLogic Auditing Provider 4-17
Security Audit Events 4-18
Adding an Auditing Provider 4-19
Configuring the Auditing Provider 4-20
Configuration Audit Events 4-21
Quiz 4-22

Summary 4-24
Practice 4-1: Work with Templates from the Command Line 4-25

5 Advanced Network Configuration
Objectives 5-2
Default WebLogic Networking 5-3
Additional Networking Scenarios 5-5
Dedicate Network Interfaces to Specific Servers 5-6
Use Multiple Ports on a Single Server 5-7
Isolate Administrative Communication 5-8
Isolate Cluster Communication 5-9
Network Channels 5-10
Channel Selection 5-11
Creating a Channel 5-12
Channel Network Settings 5-14
Channel WLST Example 5-15
Monitoring Channels 5-16
Creating Administration Channels 5-17
Administration Port 5-18
Administration Port Override 5-19
Server Standby Mode 5-20
Quiz 5-21
Summary 5-23
Practice 5-1: Use Network Channels 5-24

6 Multi Data Sources
Objectives 6-2
Data Source Review 6-3
XA Data Source Review 6-4
Multi Data Sources 6-5
Multi Data Source Architecture 6-6
Failover Option 6-7
Load Balancing Option 6-8
Connection Testing 6-9
Creating a Multi Data Source 6-10
Configuring a Multi Data Source 6-12
Multi Data Source WLST Example 6-13
Managing Multi Data Source Members 6-14
Oracle Real Application Clusters (RAC) Overview 6-15
Oracle GridLink for RAC 6-16
RAC Services 6-17

WLS and RAC Services 6-18
Creating Data Sources to Support RAC Services 6-20
Additional RAC Considerations 6-21
Quiz 6-23
Summary 6-25
Practice 6-1: Use a Multi Data Source for Failover 6-26

7 JDBC Performance Essentials
Objectives 7-2
JDBC and Application Design 7-3
Connection Pooling 7-4
Statement Caching 7-5
Connection Pinned to Thread 7-6
Logging Last Resource (LLR) Transactions 7-7
LLR Example 7-8
Configuring LLR 7-9
Quiz 7-10
Summary 7-12

8 JMS Message Management
Objectives 8-2
WebLogic JMS Review 8-3
Destination Management Features 8-5
Viewing Messages 8-6
Message Contents 8-8
Message States 8-9
Publishing a Test Message 8-10
Moving Messages 8-11
Exporting Messages to XML 8-12
Exported Messages Format 8-13
Pausing Destinations 8-14
Pausing a Destination 8-15
JMS Management WLST Examples 8-16
JMS Logging 8-17
Configuring Destination Logging 8-18
Quiz 8-19
Summary 8-21

9 JMS Guaranteed Messaging
Objectives 9-2
Guaranteed Messaging Concepts 9-3

Persistent Messaging 9-5
Default Persistent Store 9-6
Comparing File and JDBC Stores 9-7
Creating a File Store 9-8
Creating a JDBC Store 9-9
JDBC Store Details 9-10
Assigning a Store to a JMS Server 9-11
Persistent Store WLST Example 9-12
Connection Factory and Destination Persistence 9-13
Topics and Durable Subscribers 9-14
Connection Factory Client ID 9-15
Monitoring Durable Subscribers 9-16
Message Paging 9-17
Configuring Message Paging 9-18
Quiz 9-19
Summary 9-21
Practice 9-1: Configure JMS Persistence 9-22

10 JMS Performance Essentials
Objectives 10-2
JMS and Application Design 10-3
JMS Quotas Review 10-5
Configuring a JMS Server Quota 10-6
Creating a Destination Quota 10-7
Send Timeout 10-8
Quota Blocking Policies 10-9
Thresholds and Flow Control Review 10-10
Configuring Thresholds 10-11
Message Compression 10-12
Scan Expiration Interval 10-13
Message Ordering 10-14
Unit of Order (UOO) 10-15
Connection Factory UOO 10-16
Quiz 10-17
Summary 10-19

11 JMS Store and Forward
Objectives 11-2
WebLogic Store and Forward (SAF) 11-3
SAF Architecture 11-5
Creating an SAF Agent 11-6

Configuring an SAF Agent 11-7
Creating a Remote SAF Context 11-9
Creating an SAF Error Handler 11-10
SAF Imported Destinations 11-11
Creating SAF Imported Destinations 11-12
Adding Remote Endpoints 11-14
Configuring Remote Endpoints 11-15
Store and Forward WLST Example 11-16
Managing SAF Agents 11-17
Managing an SAF Agent 11-18
Managing SAF Agent Endpoints 11-19
Web Services Reliable Messaging (WSRM) 11-20
Quiz 11-22
Summary 11-24
Practice 11-1: Store and Forward JMS Messages 11-25

12 JMS Message Bridge
Objectives 12-2
WebLogic Server Messaging Bridge 12-3
Messaging Bridge Architecture 12-4
Bridging Scenarios 12-5
Comparing Bridges to Store and Forward 12-6
Resource Adapter Review 12-7
Bridge Adapters 12-8
JMS Bridge Destinations 12-10
Creating a JMS Bridge Destination 12-11
Creating a Message Bridge 12-13
Configuring a Message Bridge 12-16
Message Bridge WLST Example 12-18
Oracle Advanced Queuing (AQ) Overview 12-19
AQ Integration Overview 12-20
Creating a Foreign Server for AQ 12-21
Quiz 12-22
Summary 12-24
Practice 12-1: Bridge JMS Providers 12-25

13 Server Migration
Objectives 13-2
WebLogic Clustering Review 13-3
Whole-Server Migration 13-4

Node Manager Review 13-5
Leasing Service 13-7
Automatic Server Migration Architecture: No Failure 13-8
Automatic Server Migration Architecture: Machine Failure 13-9
Leasing Types 13-10
Configuration Overview 13-11
Network Considerations 13-12
Node Manager Network Configuration 13-13
Configuring Cluster Leasing 13-14
Database Leasing Schema 13-15
Enabling Automatic Migration for a Server 13-16
Candidate Machines 13-17
Machine Failback 13-18
Manual Server Migration 13-19
Migrating a Server 13-20
Additional File System Considerations 13-21
WebLogic Operations Control (WLOC) Overview 13-22
WLOC Architecture 13-23
Quiz 13-24
Summary 13-26
Practice 13-1: Migrate Failed Servers 13-27

14 JMS Clustering
Objectives 14-2
Road Map 14-3
JMS Communication Review 14-4
Clustered JNDI Access 14-6
Pinned Resources 14-7
JMS Cluster Targeting 14-8
Clustered Connection Factories 14-9
Reconnect Policy 14-10
Service Migration 14-11
Migratable Targets 14-12
Default Migratable Targets 14-14
Service Migration Policies for Migratable Targets 14-15
Configuring Migratable Targets 14-16
Assigning Resources to Migratable Targets 14-18
Automatic Transaction Migration 14-19
Migrating Services Manually 14-20
Service Migration WLST Example 14-21
Pre/Post Migration Scripts 14-22

Section Summary 14-24
Practice 14-1: Configure JMS High Availability 14-25
Road Map 14-26
JMS and Scalability 14-27
Distributed Destination Architecture 14-28
Distributed Queues and Topics 14-29
Creating a Distributed Destination 14-30
Targeting a Distributed Destination 14-31
Distributed Destinations and the Configuration Wizard 14-32
Distributed Destination WLST Example 14-33
Default Load Balancing Constraints 14-34
Connection Factories and Distributed Destinations 14-36
Message-Driven Bean (MDB) Review 14-37
MDBs and Distributed Destinations 14-38
Section Summary 14-39
Quiz 14-40
Lesson Summary 14-42
Practice 14-2: Load Balance JMS Messages 14-43

15 Cross-Cluster Replication
Objectives 15-2
WebLogic Session Persistence Review 15-3
In-Memory Replication Review 15-4
Using Multiple Clusters 15-6
Cross-Cluster Replication 15-7
Cross-Domain Security 15-8
Configuring Cross-Domain Security 15-9
Metropolitan Area Network (MAN) Replication 15-10
MAN Replication Example 15-11
MAN Replication Example: Server Failover 15-12
Wide Area Network (WAN) Replication 15-13
WAN Replication Example 15-14
WAN Replication Example: Cluster Failover 15-15
Configuring Cross-Cluster Replication 15-16
Advanced MAN and WAN Settings 15-17
WAN Replication Schema 15-18
Oracle Coherence Overview 15-19
Coherence*Web Overview 15-21
Coherence*Web and WebLogic Clusters 15-22
Coherence*Web Configuration Overview 15-23
Quiz 15-24

Summary 15-26
Practice 15-1: Replicate Sessions Across Two Clusters 15-27

16 Authentication Providers
Objectives 16-2
Road Map 16-3
Security Realm Review 16-4
Security Provider Stores 16-6
Store Implementations 16-7
Configuring the RDBMS Store by Using the Configuration Wizard 16-8
Default Security Configuration 16-9
Security Customization Approaches 16-10
Creating a New Security Realm 16-11
Activating a Security Realm 16-12
Security Realm WLST Example 16-13
Authentication Provider Review 16-14
Available Authentication Providers 16-15
Multiple Authentication Providers 16-17
Control Flags 16-18
Administration Groups 16-19
Section Summary 16-20
Road Map 16-21
Lightweight Directory Access Protocol (LDAP) 16-22
LDAP Structure 16-23
LDAP Search Operations 16-24
LDAP Authentication Providers 16-25
Available LDAP Authentication Providers 16-26
Configuring an LDAP Provider: Connection 16-27
Configuring an LDAP Provider: Users 16-28
Configuring an LDAP Provider: Groups 16-29
Configuring an LDAP Provider: Subgroups 16-30
Configuring an LDAP Provider: Dynamic Groups 16-31
LDAP Provider WLST Example 16-32
LDAP Failover 16-33
LDAP Caching 16-34
LDAP X509 Identity Asserter Overview 16-35
Section Summary 16-37
Practice 16-1: Authenticate Using an External LDAP 16-38
Road Map 16-39
Database Authentication Providers 16-40
Available Database Authenticators 16-41

Authenticator Data Sources 16-42
Configuring the SQL Authenticator 16-43
Configuring the SQL Authenticator: Passwords 16-44
Configuring the SQL Authenticator: Subgroups 16-45
SQL Authenticator WLST Example 16-46
Default Schema 16-47
Section Summary 16-48
Practice 16-2: Authenticate Using a Database 16-49
Road Map 16-50
Password Validation Providers 16-51
Authentication Provider Support 16-52
Default Authenticator and Password Validation 16-53
Password Composition Rules 16-54
Accessing the Password Validator 16-55
Configuring the Password Validator 16-56
Password Validator WLST Example 16-58
Section Summary 16-59
Practice 16-3: Define Password Rules 16-60
Road Map 16-61
Security Migration Scenarios 16-62
Provider Migration Support 16-63
Migration Approaches 16-64
Migration Formats 16-65
Migration Example: XACML Authorizer 16-66
Security Migration WLST Example 16-67
Section Summary 16-68
Quiz 16-69
Lesson Summary 16-72

17 Server Performance Essentials
Objectives 17-2
Road Map 17-3
Performance Terminology 17-4
Bottlenecks 17-5
Benchmarking 17-6
Performance Testing Methodology 17-7
Load Testing 17-8
Load Testing Tools 17-9
The Grinder 17-10
The Grinder Architecture 17-11
The Grinder Proxy 17-12

Agent Properties 17-13
The Grinder Console 17-14
Section Summary 17-15
Road Map 17-16
Java Virtual Machine (JVM) Selection 17-17
Setting WLS JVM Arguments 17-18
JRockit Review 17-19
JVM Tuning Parameters 17-20
Java Heap 17-21
WLS Production Recommendations for Any JVM 17-22
Generational Garbage Collection 17-23
Basic JRockit JVM Arguments 17-25
JRockit Recommendations for WLS 17-27
Sun Hotspot JVM Generational GC 17-28
Basic Sun JVM Arguments 17-29
Sun JVM Recommendations for WLS 17-30
Monitoring a WLS JVM 17-31
WLS Low Memory Detection 17-32
Configuring Low Memory Detection 17-33
JRockit Command-Line Heap Monitoring 17-34
Sun JVM Heap Monitoring 17-35
JRockit Mission Control (JRMC) 17-36
Management Console Features 17-37
Management Console Heap Monitoring 17-38
Flight Recorder Overview 17-39
Flight Recorder Features 17-40
Flight Recorder Code Analysis 17-41
Section Summary 17-42
Practice 17-1: Tune a Server JVM 17-43
Road Map 17-44
Production Mode 17-45
Memory Chunk Settings 17-46
Logging Considerations 17-47
Log Filters Review 17-48
JavaServer Page (JSP) Review 17-49
Precompiling JSP Files 17-50
JSP and Class Reloading 17-51
Using Web Servers for Static Content 17-52
Cluster Considerations 17-53
Session Persistence Cache Size 17-54
Section Summary 17-55

Practice 17-2: Tune Server Performance 17-56
Road Map 17-57
WebLogic Server Threads 17-58
Monitoring a Server Thread Pool 17-59
Monitoring Server Threads 17-60
Stuck Thread Handling 17-61
Configuring Stuck Thread Handling 17-62
Application Stuck Thread Handling 17-64
Work Managers 17-65
Work Manager Scope 17-66
Work Manager Architecture 17-67
Request Class Types 17-68
Creating a Work Manager 17-69
Creating a Request Class 17-70
Constraint Types 17-71
Creating a Constraint 17-72
Work Manager WLST Example 17-73
Work Managers and Stuck Threads 17-74
Assigning Work Managers to Applications 17-75
Section Summary 17-76
Quiz 17-77
Lesson Summary 17-80
Practice 17-3: Tune Performance Using Work Managers 17-81

18 Monitoring and Diagnostics Essentials
Objectives 18-2
Road Map 18-3
Why Monitoring Tools? 18-4
Oracle Monitoring Tools 18-5
Java Management Extension (JMX) Review 18-6
WLS MBean Hierarchies 18-7
Monitoring with the Console 18-8
Monitoring with WLST 18-9
Runtime MBean Documentation 18-10
Fusion Middleware (FMW) Control Review 18-11
FMW Control: Viewing Farm Topology 18-12
FMW Control: Monitoring Servers 18-13
FMW Control: Monitoring Deployments 18-14
Guardian Overview 18-15
Section Summary 18-16
Road Map 18-17

WebLogic Diagnostics Framework (WLDF) 18-18
WLDF Architecture 18-19
WLDF Configuration Overview 18-20
Capturing a Server Diagnostic Image 18-21
Diagnostic Archives 18-23
Configuring Server Diagnostic Archives 18-24
Creating a Diagnostic Module 18-25
Metric Collectors 18-26
Configuring a Metric Collector 18-27
Watches and Notifications 18-28
Configuring a Watch 18-29
WLDF WLST Examples 18-31
Sample WLDF Framework 18-32
Instrumentation 18-34
Working with Diagnostic Monitors 18-36
Configuring a System-Scoped Monitor 18-37
WLDF Monitoring Dashboard 18-39
Views, Charts, and Graphs 18-40
Anatomy of a Chart 18-41
Creating a New Graph 18-42
Section Summary 18-43
Practice 18-1 Configure and Monitor Diagnostic Data 18-44
Road Map 18-45
Simple Network Management Protocol (SNMP) 18-46
SNMP Architecture 18-47
Object Identifier (OID) 18-48
Management Information Base (MIB) 18-49
WLS MIB and OIDs 18-50
Common SNMP Message Types 18-51
WLS SNMP Architecture 18-52
Creating an SNMP Agent 18-54
Configuring an SNMP Agent 18-55
SNMP Channels 18-57
WLS SNMP Notifications 18-58
Creating Trap Monitors 18-59
Creating Trap Destinations 18-60
SNMP Security 18-61
Configuring Agent Security 18-62
Configuring SNMPv3 Credentials 18-63
Configuring Trap Destination Security 18-64
WLS SNMP Utility 18-65

Section Summary 18-66
Practice 18-2 Monitor WLS Using SNMP 18-67
Road Map 18-68
Subsystem Debugging 18-69
Debug Scopes 18-70
Example Debug Scopes and Attributes 18-71
Debug Logging 18-73
Section Summary 18-74
Quiz 18-75
Lesson Summary 18-79
Practice 18-3 Debug WLS Subsystems 18-80

15

Cross-Cluster Replication

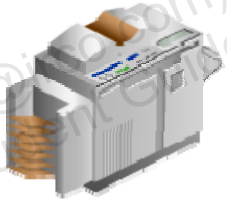
Objectives

After completing this lesson, you will be able to:

- Describe the process of in-memory session replication
- Configure cross-cluster session replication over a metropolitan area network (MAN) or wide area network (WAN)
- Initialize a database to support WAN replication
- Explain how Oracle Coherence may be used to provide session availability

WebLogic Session Persistence Review

- Web applications use HTTP sessions to track information in server memory for each client.
- By default, when a client fails over to another server in the cluster, its session information is lost.
- WebLogic Server supports several session persistence strategies to recover sessions when clients are redirected from failed servers, including:
  - In-memory replication
  - JDBC persistence
  - File persistence
  - Cookie persistence
- Persistence can be done synchronously or asynchronously.



WebLogic Session Persistence Review

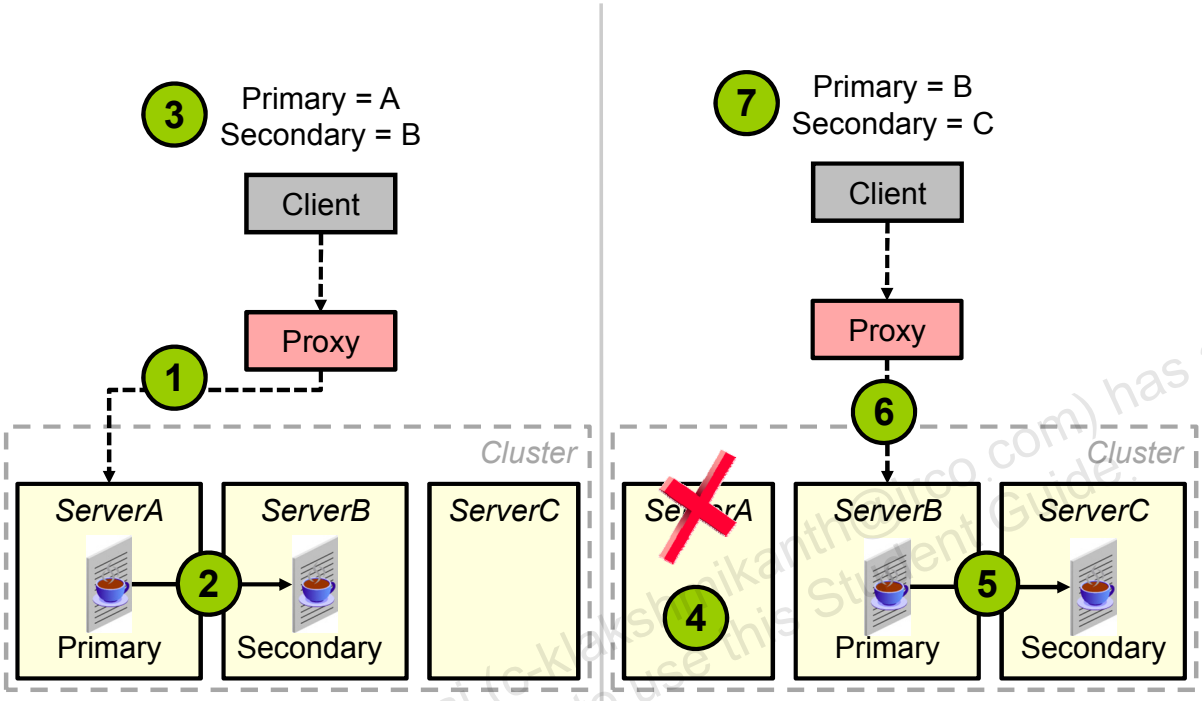
Web application components, such as servlets and JavaServer Pages (JSPs), maintain data on behalf of clients using an `HttpSession` instance that is available on a per-client basis. To provide high availability of Web applications, shared access to one `HttpSession` object must be provided. `HttpSession` objects can be replicated within WebLogic Server by storing their data using in-memory replication or file system persistence, or by storing it in a database.

For clusters that use a supported hardware load-balancing solution, the load-balancing hardware redirects client requests to any available server in the WebLogic Server cluster. The cluster itself obtains the replica of the client's HTTP session state from a secondary server in the cluster.

In clusters that use Web servers with WebLogic proxy plug-ins, the proxy plug-in handles failover transparently to the client. If a server fails, the plug-in locates the replicated HTTP session state on a secondary server and redirects the client's request accordingly.

With synchronous replication, the secondary is updated at the end of the current request and before the response is sent back to the client. This helps to guarantee reliability and data integrity. Alternatively, to achieve greater performance, replication can instead be performed asynchronously. With this approach, session updates are placed on an internal queue, and when the queue is full, all changes are "flushed" to the secondary.

In-Memory Replication Review



In-Memory Replication Review

The graphic in the slide depicts a client accessing a Web application hosted in a cluster. All client requests are forwarded to the WebLogic Server cluster via a Web server plug-in:

1. The proxy load balances a client's initial request to Server A. The Web application creates a new session for this user.
2. To provide failover services for the Web application, the primary server replicates the client's session state to a secondary WebLogic Server in the cluster. This ensures that a replica of the session state exists even if the primary server fails (for example, due to a network failure). In the example in the slide, Server B is selected as the secondary server.
3. The HTTP response back from Server A includes a cookie indicating that it is the primary server and Server B is the secondary.

A typical failure scenario is as follows:

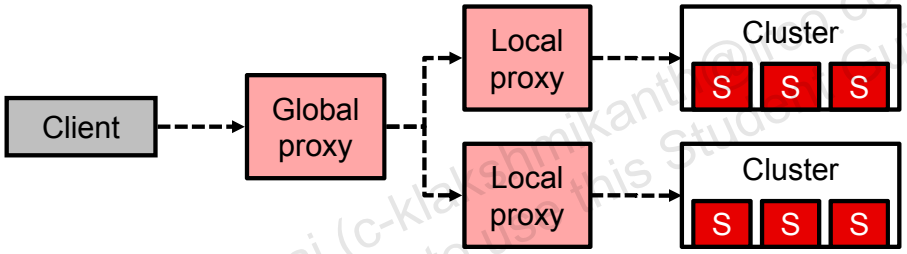
4. The primary server for the user fails.
5. After the failure, Server B becomes the primary server hosting the session state, and a new secondary is created (Server C, in this example).

In-Memory Replication Review (continued)

6. On the next request from the client, the proxy uses the client's cookie information to determine the location of the secondary WebLogic Server that hosts the replica of the session state. The proxy automatically redirects the client's next HTTP request to the secondary server, and the failover is transparent to the client.
7. In the HTTP response, the proxy updates the client's cookie to reflect the new primary and secondary servers to account for the possibility of subsequent failovers.

Using Multiple Clusters

- Due to unicast and multicast communications, a cluster is typically limited to a single local area network (LAN).
- Using multiple clusters allows you to:
  - Distribute your application across geographic areas and service providers
  - Achieve a higher degree of fault tolerance
- This architecture requires a global proxy that can direct clients back to their host cluster (cluster affinity).



Using Multiple Clusters

In addition to providing HTTP session state replication across servers within a cluster, WebLogic server provides the ability to replicate HTTP session state across multiple clusters. This improves high-availability and fault tolerance by allowing clusters to be spread across multiple geographic regions, power grids, and Internet service providers (ISPs).

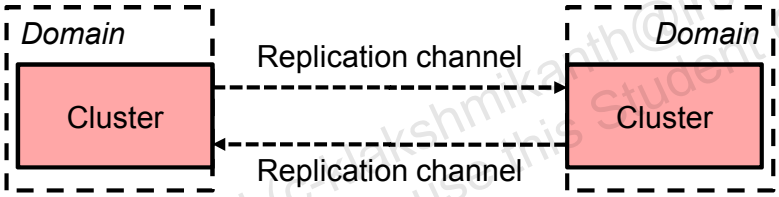
To perform cross-cluster replication with WebLogic Server, your network must include global and local load balancers. These can be hardware and/or Web server load balancers. In a network configuration that supports cross-cluster replication, the global load balancer is responsible for balancing HTTP requests across clusters. When a request comes in, the global load balancer determines which cluster to send it to based on the current number of requests being handled by each cluster. Then the request is passed to the local load balancer for the chosen cluster. The local load balancer receives HTTP requests from the global load balancer. The local load balancer is responsible for balancing HTTP requests across servers within the cluster.

When a server within a cluster fails, the local load balancer is responsible for transferring the request to other servers within a cluster. When the entire cluster fails, the local load balancer returns HTTP requests back to the global load balancer. The global load balancer then redirects this request to the other local load balancer.

Cross-Cluster Replication

WebLogic Server supports session replication across two clusters, provided they:

- Are in separate domains, each with a unique name
- Have identical configurations
- Have the same number of servers
- Are configured to trust internal messages from each other
- Communicate with each other using the default network channel or a dedicated custom channel.



Cross-Cluster Replication

Each domain should be set up and configured identically. In addition to identical domain, cluster and server configuration, the number of servers, clusters, and so on should be identical. Similarly, application deployment should be identical in each domain.

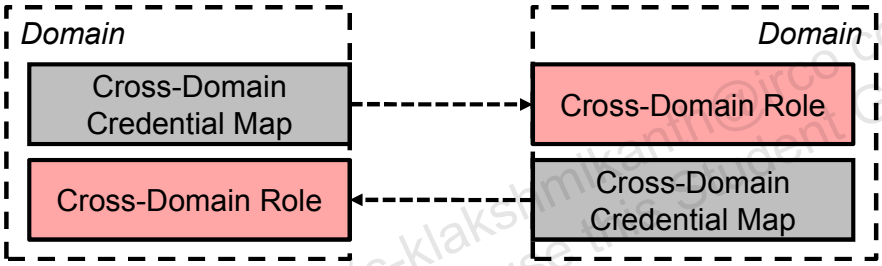
You must configure your load balancer to maintain session IDs. If the load balancers do not maintain session ID, subsequent requests will always be sent to a new server. You must also configure your load balancer to know which backup cluster to use when a primary cluster or server fails.

Cross-domain security establishes trust between domains such that dedicated internal users or principals from one domain can send messages to another domain. Various WebLogic Server subsystems can be configured to communicate with another domain in a secure fashion, including the cross-cluster session replication features.

The specific method used to replicate session information depends on which type of cross-cluster replication you are implementing. By default, clusters replicate session data between their servers using the servers' default network channels. Alternatively, you can configure a custom network channel on each server in a cluster and dedicate it to cross-cluster communication. Each server in the cluster must use the same channel name.

Cross-Domain Security

- When enabled, a domain will only accept or “trust” internal server requests from other specific domains.
- Add a user to the `CrossDomainConnector` role (`CrossDomainConnectors` group, by default).
- Specify the credentials to use for outgoing, cross-domain traffic.

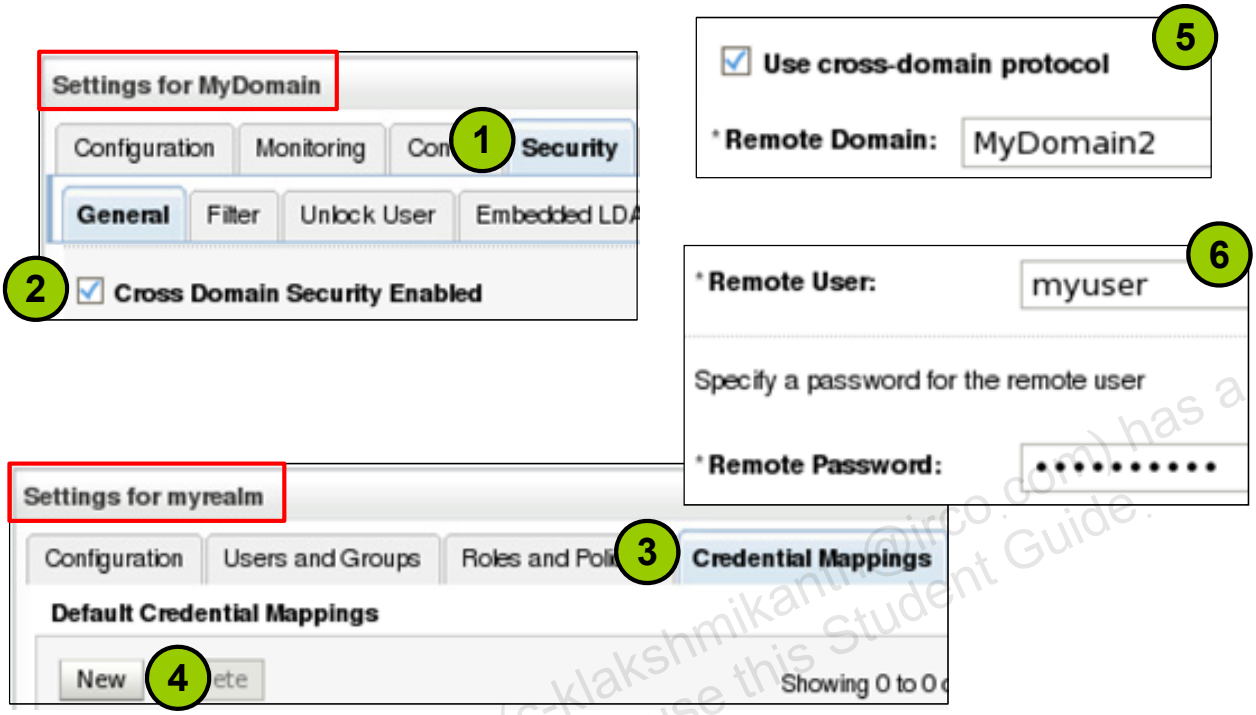


Cross-Domain Security

Cross-domain security establishes trust between two WebLogic Server domains by using credential mappings to secure their internal communications. Credential mappings specify the credentials (username/password) to include when sending internal messages to other domains. The Credential Mapper identifies domains by their names. Therefore, it is important that the domains involved have unique names.

These credentials are then authorized against a global security role named CrossDomainConnector, which by default is associated with a group named CrossDomainConnectors. Also, by default the CrossDomainConnectors group has no users as members. You need to create one or more users and add them to the group. Typically, such a user will be a virtual system user and preferably should have no privileges other than those granted by the CrossDomainConnector security role.

Configuring Cross-Domain Security



Configuring Cross-Domain Security

1. Click the name of the domain in the Domain Structure panel. Then click the Security tab.
2. Select the check box Cross Domain Security Enabled and click Save.
3. In the Domain Structure panel, click Security Realms. After selecting your security realm, click the Credential Mappings tab.
4. Click New.
5. Select the “Use cross-domain protocol” check box. Then enter the name of the Remote Domain. Click Next.
6. Enter the remote user and remote password to use with the selected remote domain. This user must be a member of the domain’s CrossDomainConnector role. Click Finish.

Metropolitan Area Network (MAN) Replication

- Synchronously replicates sessions in-memory to the opposite cluster
- Assumes a low-latency network between clusters
- Assumes that users are pinned to a cluster unless the entire cluster fails

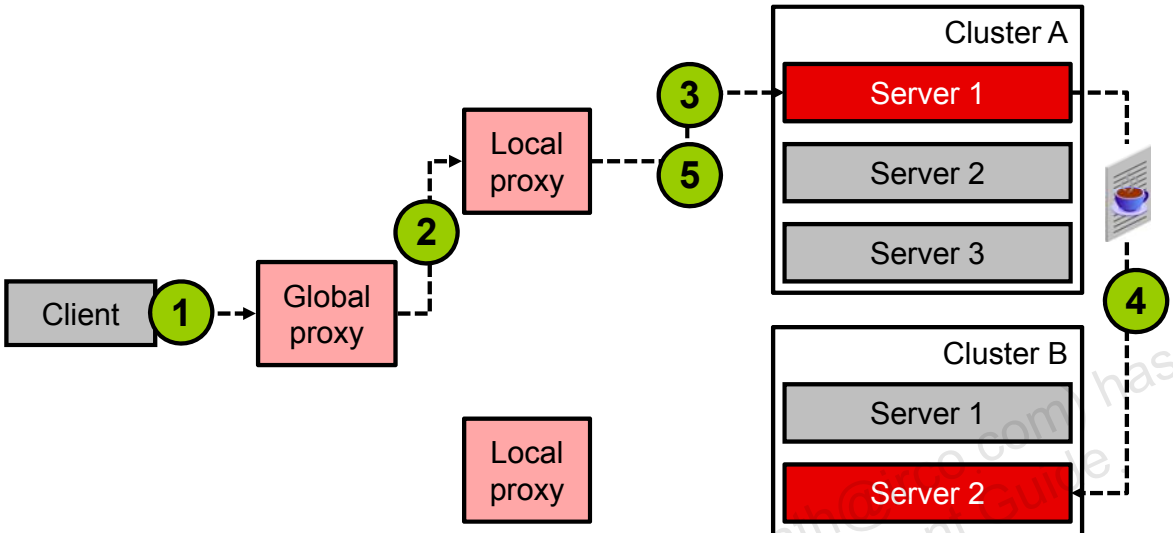
Metropolitan Area Network (MAN) Replication

Resources within a metropolitan area network (MAN) are often in physically separate locations, but are geographically close enough that network latency is not an issue. Network communication in a MAN generally has low latency and fast interconnect. Clusters within a MAN can be installed in physically separate locations, which improves availability.

To provide failover within a MAN, WebLogic Server provides an in-memory mechanism that works between two separate clusters. This allows session state to be replicated synchronously from one cluster to another, provided that the network latency is a few milliseconds. The advantage of using a synchronous method is that reliability of in-memory replication is guaranteed. The performance of synchronous state replication is dependant on the network latency between clusters. You should use this method only if the network latency between the clusters is tolerable.

MAN replication relies on global load balancers to maintain cluster affinity and local load balancers to maintain server affinity. If a server within a cluster fails, the local load balancer is responsible for ensuring that session state is replicated to another server in the cluster. If all of the servers within a cluster have failed or are unavailable, the global load balancer is responsible for replicating session state to another cluster. After a client establishes a connection through a load balancer to a cluster, the client must be pinned to that cluster as long as it is healthy.

MAN Replication Example

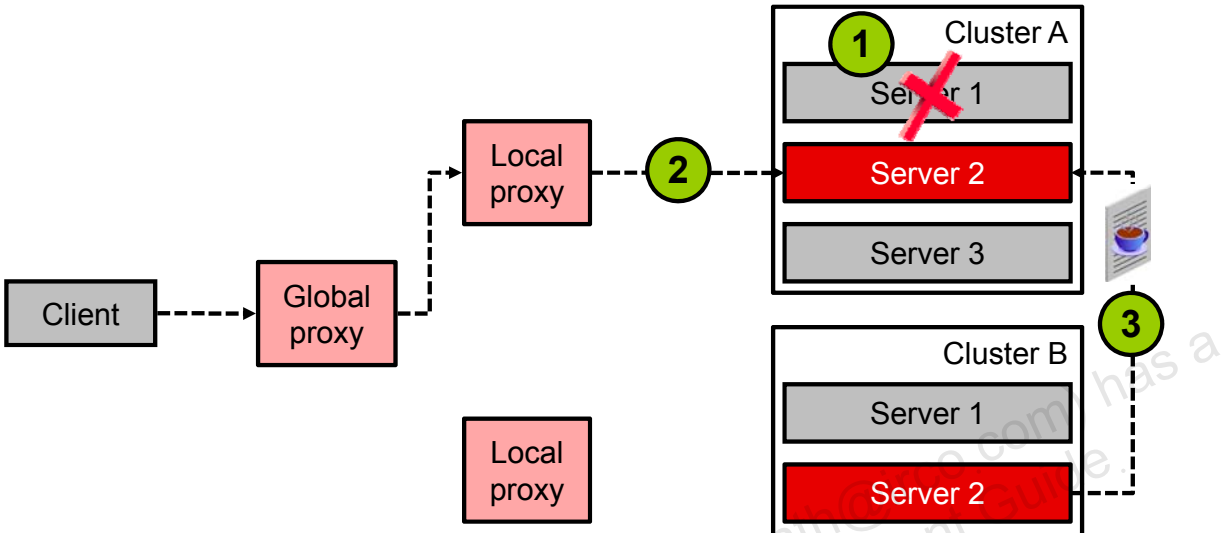


MAN Replication Example

A typical interaction with a MAN cross-cluster configuration is as follows:

1. A client makes an initial request to a Web application.
2. The request is directed to the global proxy, which in turn load balances the client to one of the local proxies for each cluster.
3. Similarly, the local proxy load balances the new client to one of the servers in its cluster, in this case Server 1 on Cluster A. The Web application creates a new session for this user.
4. To provide failover services for the Web application, the primary server replicates the client's session state to a secondary server in the remote cluster, Cluster B. In this example, Server 2 was selected to be the secondary.
5. The HTTP response back from Server 1 includes a cookie indicating that it is the primary server. Consequently, the global and local proxies pin subsequent requests from this client to Server 1. A cookie also indicates that Server 2 on Cluster B is the secondary, but this information is not used by the proxy.

MAN Replication Example: Server Failover



MAN Replication Example: Server Failover

A server failure scenario for MAN cross-cluster replication is described as follows:

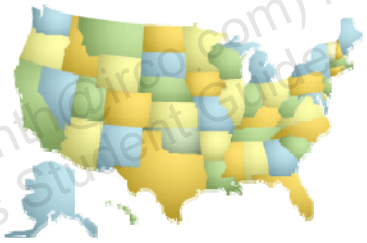
1. Server 1 fails.
2. Because the client's primary server has failed, the local proxy for Cluster A must direct subsequent requests to another server in the cluster. In this example, Server 2 is selected.
3. Server 2 on Cluster A does not have a copy of this client's session data. However, using the client's cookie, it determines that Server 2 on Cluster B is this client's secondary. Therefore, it contacts Cluster B and requests a copy of the session. As always, the server also updates the cookie to reflect that it has become the new primary.

If all of the servers in Cluster A fail, the global load balancer will automatically fail all subsequent session requests to Cluster B. All sessions that have been replicated to Cluster B will be recovered and the client will experience no data loss.

If the secondary server fails, then the primary server will automatically select a new secondary server on Cluster B. Upon receiving a client request, the session information will be backed up on the new secondary server.

Wide Area Network (WAN) Replication

- Assumes a high latency network between clusters
- Continues to use synchronous in-memory replication within the same cluster
- Also asynchronously replicates sessions to a database shared by both clusters
- Requires a Java Database Connectivity (JDBC) data source

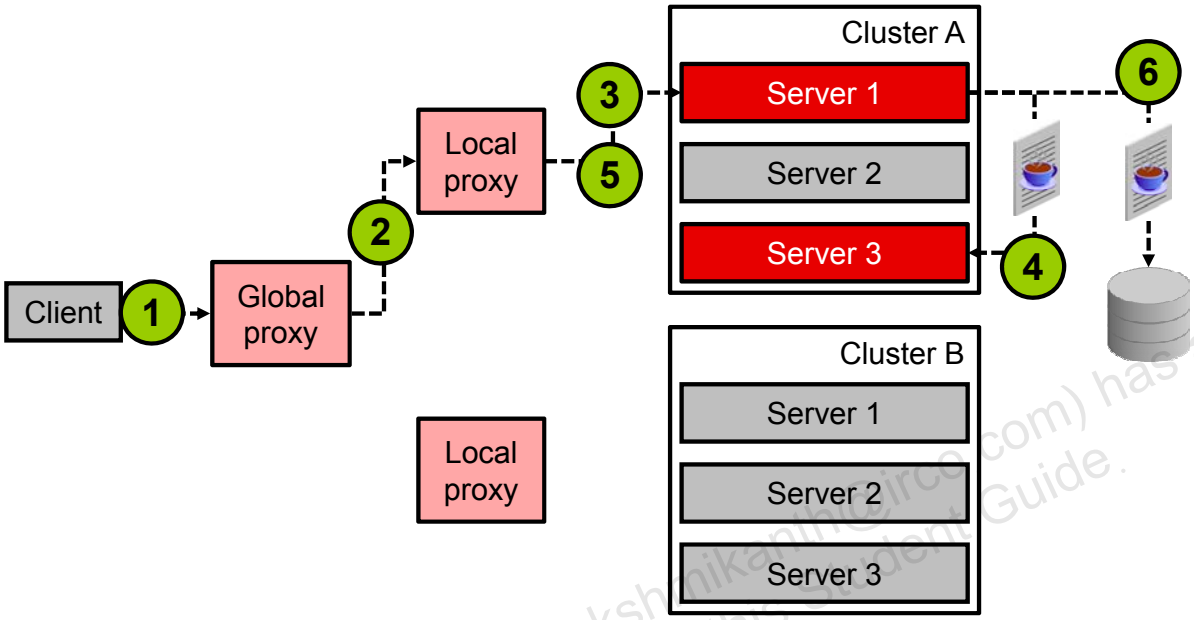


Wide Area Network (WAN) Replication

Resources in a wide area network (WAN) are frequently spread across separate geographical regions. In addition to requiring network traffic to cross long distances, these resources are often separated by multiple routers and other network bottlenecks. Network communication in a WAN generally has higher latency and slower interconnect. Slower network performance within a WAN makes it difficult to use a synchronous replication mechanism like the one used within a MAN. WebLogic Server provides failover across clusters in WAN by using an asynchronous data replication scheme.

To enable cross-cluster replication within a WAN environment, you must create a JDBC data source that points to the database where session state information is stored. Create this data source on both domains and target it to both clusters.

WAN Replication Example

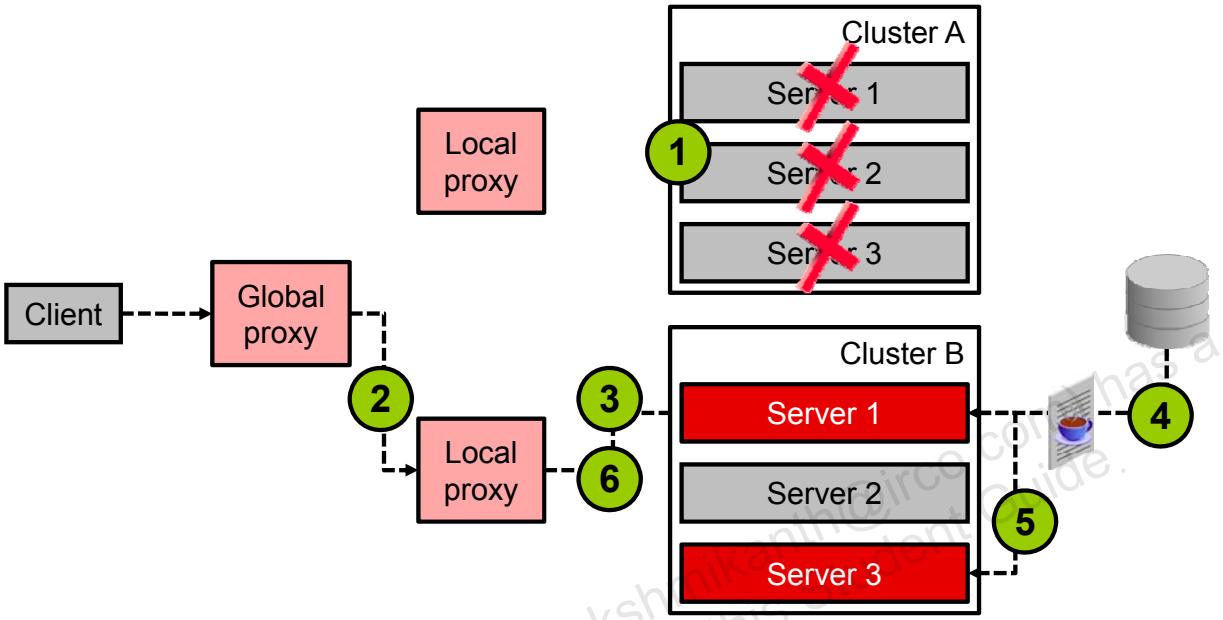


WAN Replication Example

A typical interaction with a WAN cross-cluster configuration is described as follows:

1. A client makes an initial request to a Web application.
2. The request is directed to the global proxy, which in turn load balances the client to one of the local proxies for each cluster.
3. Similarly, the local proxy load balances the new client to one of the servers in its cluster, in this case, Server 1 on Cluster A. The Web application creates a new session for this user.
4. To provide failover services for the Web application, the primary server replicates the client's session state to a secondary server in the same cluster. In this example, Server 3 was selected to be the secondary.
5. The HTTP response back from Server 1 includes a cookie indicating that it is the primary server and Server 3 is the secondary.
6. While the client continues to interact with the Web application, Server 1 asynchronously persists the latest session changes to a highly available database that is accessible to both clusters. This database will be used to provide failover if the entire cluster fails.

WAN Replication Example: Cluster Failover

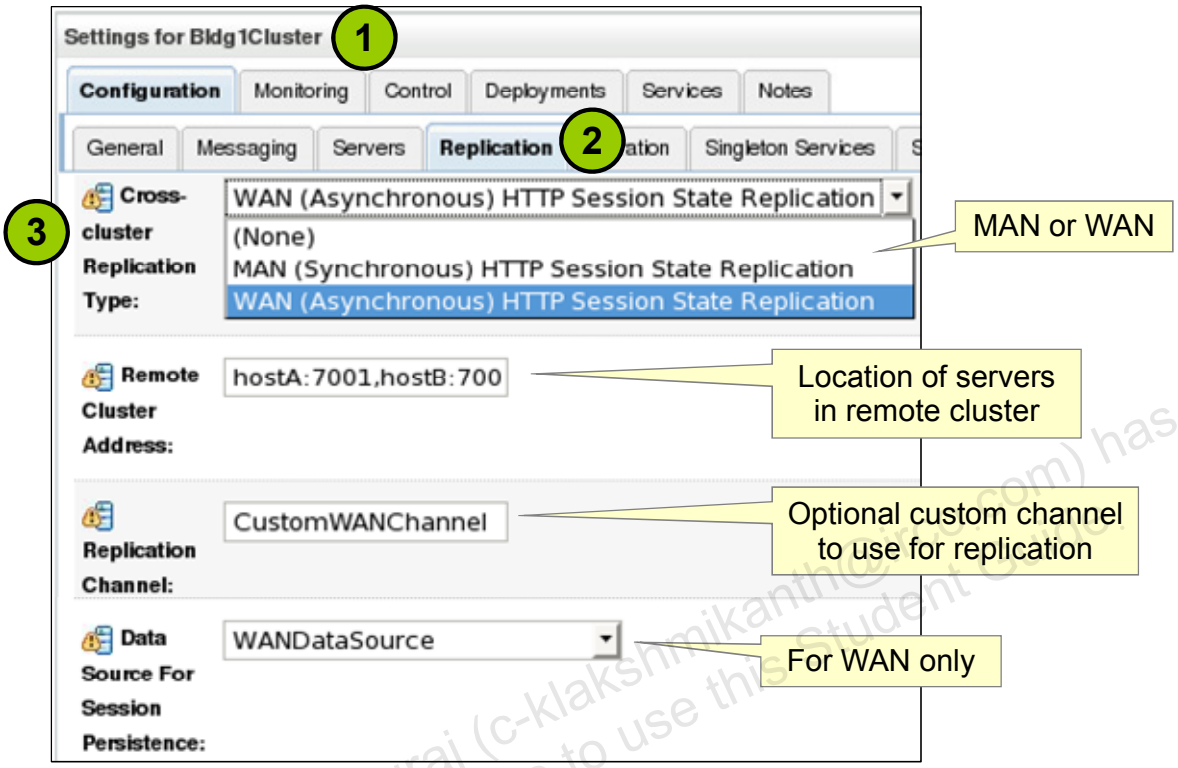


WAN Replication Example: Cluster Failover

A cluster failure scenario for WAN cross-cluster replication is described as follows:

1. All of the servers in Cluster A fail.
2. The request is sent to the global proxy, which now has no choice but to direct the request to the local proxy for Cluster B.
3. The local proxy selects an available server in Cluster B. In this example, Server 1 is chosen.
4. Server 1 has already detected that Cluster A is unavailable. Also, because of the presence of a cookie indicating primary and secondary servers in Cluster A, Server 1 retrieves the user's persisted session from the database shared by both clusters.
5. Server 1 selects another server in the same cluster to be the new secondary. In this example, Server 3 is chosen.
6. The HTTP response back from Server 1 includes an updated cookie indicating that it is the primary server and Server 3 is the secondary.

Configuring Cross-Cluster Replication



Configuring Cross-Cluster Replication

Before configuring cross-cluster replication using the administration console, you must set up your network environment, including hardware load balancers, WebLogic Server instances, and clusters. If you are using a cross-cluster replication within a WAN environment, you must also create and configure a data source.

1. Select an existing cluster.
2. Click Configuration > Replication.
3. Select a value for “Cross-cluster Replication Type” (WAN or MAN).
4. Enter values for the following:
   - **Remote Cluster Address:** Addresses used to communicate with the secondary cluster. For multiple addresses, use a comma as the delimiter.
   - **Replication Channel:** The name of the existing network channel used to communicate with the secondary cluster. This network channel should be targeted to the entire cluster. Both these settings should be configured so that communication between the primary and secondary cluster is not routed through the hardware load balancers.
5. If you configure cross-cluster replication in a WAN environment, select a value for “Data Source for Session Persistence.”

Advanced MAN and WAN Settings

Field	MAN or WAN	Description
Persist Sessions on Shutdown	WAN	All sessions are synchronized with the database when servers receive a shutdown request.
Secure Replication Enabled	Both	If no custom channel is specified, should sessions be replicated using SSL and default secure port
Session Flush Interval	WAN	How often (in seconds) the current batch of session updates should be written to the database
Session Flush Threshold	WAN	The number of session updates after which they are written to the database as a batch
Inter-Cluster Comm Link Health Interval Check	Both	After a remote cluster becomes inaccessible, how often (in milliseconds) should this cluster check to see if it has become available again

WAN Replication Schema

WAN replication requires a slightly different database schema than standard session persistence.

```
CREATE TABLE WLS_WAN_PERSISTENCE_TABLE (
  WL_ID VARCHAR2(100) NOT NULL,
  WL_CONTEXT_PATH VARCHAR2(50) NOT NULL,
  WL_CREATE_TIME NUMBER(20),
  WL_ACCESS_TIME NUMBER(20),
  WL_MAX_INACTIVE_INTERVAL NUMBER(38),
  WL_VERSION NUMBER(20) NOT NULL,
  WL_INTERNAL_ATTRIBUTE NUMBER(38),
  WL_SESSION_ATTRIBUTE_KEY VARCHAR2(100),
  WL_SESSION_ATTRIBUTE_VALUE LONG RAW,
  PRIMARY KEY(WL_ID, WL_CONTEXT_PATH,
  WL_VERSION, WL_SESSION_ATTRIBUTE_KEY));
```


WAN Replication Schema

- **WL_ID:** Stores the HTTP session ID
- **WL_CONTEXT_PATH:** Stores the context path to the Web application that created the session
- **WL_CREATE_TIME:** Stores the time the session state was created
- **WL_ACCESS_TIME:** Stores the time of the last update to the session state
- **WL_VERSION:** Stores the version of the session. Each update to a session has an associated version.

Oracle Coherence Overview

Coherence:

- Provides a distributed, in-memory caching solution for Java
- Is based on a *grid* of cache servers, or *nodes*
- Automatically distributes or partitions cached data across the grid based on the number of servers
- Implements replication for high availability
- Includes a decentralized management and monitoring model based on JMX
- Supports a proxy architecture to span multiple LANs
- Supports additional languages such as .NET and C++

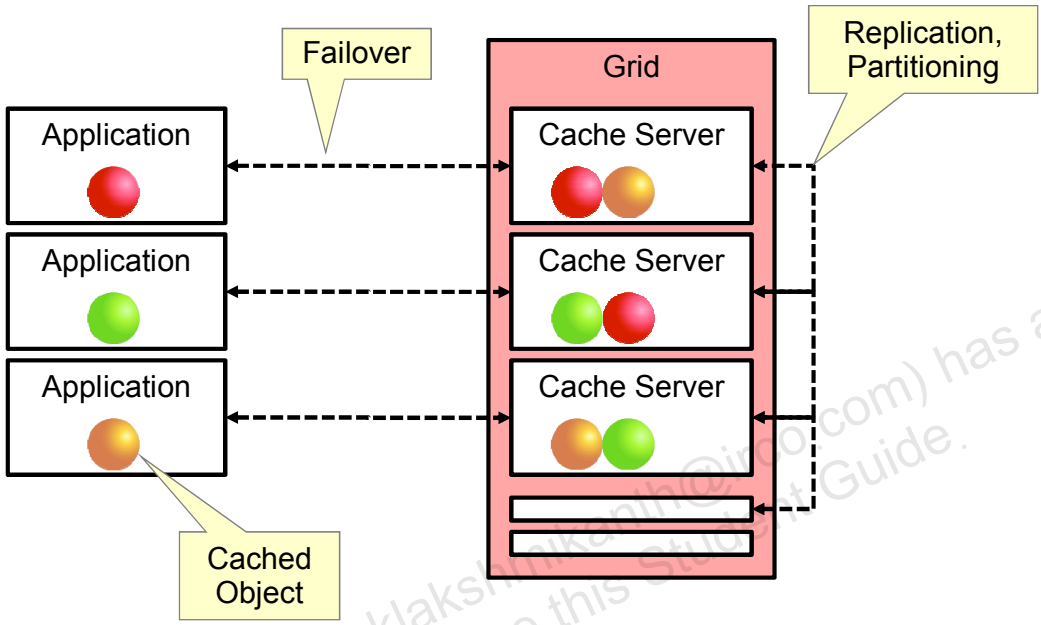
Oracle Coherence Overview

One of the primary uses of Oracle Coherence is to cluster an application's objects and data. In the simplest sense, this means that all the objects and data that an application delegates to Coherence are automatically available to and accessible by all servers in the application cluster. None of the objects or data will be lost in the event of server failure. By clustering the application's objects and data, Coherence solves many of the difficult problems related to achieving availability, reliability, scalability, performance, serviceability, and manageability of clustered applications.

Oracle Coherence is a JCache-compliant, in-memory caching and data management solution for clustered Java EE applications and application servers. Coherence makes sharing and managing data in a cluster as simple as on a single server. It accomplishes this by coordinating updates to the cached data by using cluster-wide concurrency control, replicating and distributing data modifications across the cluster, and delivering notifications of data modifications to any servers that request them. Developers can easily take advantage of Coherence features by using the standard Java Collections API to access and modify data, and by using the standard JavaBean event model to receive data change notifications.

Coherence provides a cluster-wide view of management information through the standard JMX API, so that the entire cluster can be managed from a single server. The information provided includes cache sizes along with hit and miss rates.

Oracle Coherence Overview



Oracle Coherence Overview (continued)

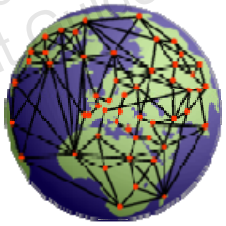
Partitioning refers to the ability of Coherence to load-balance data storage, access and management across all of the servers in the cluster. For example, when using Coherence data partitioning, if there are four servers in a cluster, each will manage 25% of the data. And if another server is added, each server will dynamically adjust so that the five servers will manage 20% of the data. This data load balancing will occur without any application interruption and without any lost data or operations. Similarly, if one of those five servers were to fail, each of the remaining four servers would readjust to managing 25% of the data. Once again, there is no data loss, including the 20% of the data that was being managed on the failed server.

While the partitioning feature dynamically load balances data evenly across the entire server cluster, replication ensures that a desired set of data is always available and up-to-date at all times in the cluster. Replication allows operations running on any server to obtain the data that they need locally, at basically no cost, because that data has already been replicated to that server. The only downside of partitioning is that it introduces latency for data access, and in most applications the data access rate far outweighs the data modification rate. To eliminate the latency associated with partitioned data access, Coherence can employ “near caching.” Frequently and recently used data from the partitioned cache are maintained on the specific servers that are accessing that data, and this near data is kept up-to-date using event-based invalidation.

Coherence*Web Overview

Coherence*Web:

- Is a plug-in that enhances WebLogic Server's session management implementation with a Coherence grid
- Provides a highly scalable session replication solution that can span WebLogic domains and clusters
- Reduces WebLogic Server's memory footprint
- Requires no changes to existing application code
- Allows session data to be shared across applications
- Supports locking to prevent concurrent access
- Can be enabled on an application basis



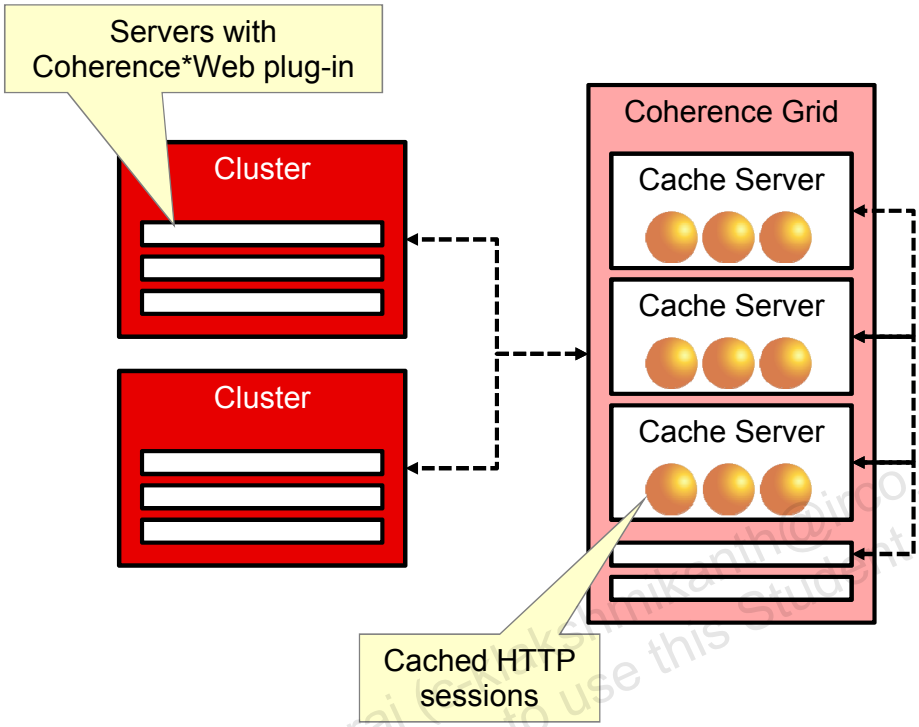
Coherence*Web Overview

Coherence*Web is an application server plug-in dedicated to managing session state in clustered environments. It brings the scalability, availability, reliability, and performance characteristics of a Coherence data grid to in-memory session management and storage. Also, because it is not constrained by the deployment topologies of the application server, it enables session sharing and management across different Web applications, domains, and even different application server products. Sometimes you may want to explicitly prevent session data from being shared by different JavaEE applications that participate in the same Coherence cluster, so Coherence*Web supports this approach as well.

In the case where sessions are shared across Web applications, you may want to scope individual session attributes so that they are either globally visible (that is, all Web applications can see and modify these attributes) or scoped to an individual Web application (that is, not visible to any instance of another application). The latter approach may be desired to help avoid namespace collisions when multiple applications use the same session attribute names.

With Coherence*Web, session data is stored outside of the application server, thereby freeing server heap space. This architecture also allows you to individually tune and scale the sizes of your application server clusters and session data grids.

Coherence*Web and WebLogic Clusters



Coherence*Web and WebLogic Clusters

Coherence*Web is not a replacement for WebLogic Server's in-memory HTTP state replication services. However, Coherence*Web should be considered when an application has large HTTP session state objects, when running into memory constraints due to storing HTTP session object data, or if you have an existing Coherence cluster and want to off-load HTTP Session storage to a Coherence cluster.

The Coherence*Web SPI for WebLogic Server is configured with local-storage disabled. This means a Coherence cache server must be running in its own JVM, separate from the JVM running WebLogic Server.

By default, Coherence*Web creates a single HTTP Session across all Web applications for each client and scopes the session attributes to each Web application. This means that if a session is invalidated in one Web application, that same session is invalidated for all Web applications in WebLogic Server using Coherence*Web. If you do not want this behavior, a potential work-around is to edit the `<cookie-path>` element of the `weblogic.xml` descriptor.

Coherence*Web Configuration Overview

1. Apply any necessary patches.
2. Start one or more Coherence cache servers on your network.
3. Deploy `coherence-web-spi.war` as a shared library on each WebLogic Server.
4. Add `coherence.jar` to participating applications.
5. Update each application's `weblogic.xml` descriptor file and add a reference to the `coherence-web-api` library.

```
<library-ref>
  <library-name>coherence-web-spi</library-name>
  <specification-version>1.0.0.0</specification-version>
  <implementation-version>1.0.0.0</implementation-version>
  <exact-match>false</exact-match>
</library-ref>
```


Coherence*Web Configuration Overview

The WebLogic Server Coherence*Web SPI consists of the coherence-web-spi.war file, located in the `coherence/lib` directory in the Coherence distribution. This file is a deployable shared library that contains a native plug-in to WebLogic Server's HTTP session management interface.

The `coherence.jar` file, located in the same `coherence/lib` directory, is also necessary for enabling Coherence*Web functionality in WebLogic Server. It can be packaged within an EAR file at `APP-INF/lib` or a WAR file at `WEB-INF/lib`. It is recommended that you use this approach over copying the file to the `/lib` folder of your domain.

The Coherence caches used by the Coherence*Web SPI are configured by the `session-cache-config.xml` file. This file is located inside the `coherence-web-spi.war` file under the `WEB-INF\classes` directory. Although the default configuration should be sufficient for most deployments, it can be customized as needed.

Because Coherence*Web is in control of the HTTP session life cycle, most standard parameters for the `<session-descriptor>` element in either `weblogic.xml` or `weblogic-application.xml` are ignored. However, Coherence*Web does support several `<context-param>` parameters to customize the plug-in's behavior. Refer to the documentation for details.

Quiz

Which types of WebLogic Server session persistence require a database?

a. Memory
b. MAN
c. WAN
d. Cookie
e. JDBC

Answer: c, e

Quiz

Which of these is a required cluster attribute for cross-cluster replication?

a. Remote Cluster Address
b. Migration Basis
c. Client ID
d. Candidate Machines

Answer: a

Summary

In this lesson, you should have learned how to:

- Describe how in-memory replication works within a single cluster
- Compare and contrast MAN and WAN session replication
- Configure a cluster to use MAN or WAN replication
- List some of the capabilities of Oracle Coherence
- Describe an alternative session replication approach using Coherence*Web

Practice 15-1:
Replicate Sessions Across Two Clusters

This practice covers the following topics:

- Starting additional load balancers
- Creating two new domains, each with a cluster of two servers
- Configuring clusters for MAN session replication
- Verifying failover after primary cluster failure

16

Authentication Providers

ORACLE

Objectives

After completing this lesson, you will be able to:

- Describe WebLogic's authentication architecture
- Create a security realm
- Configure several types of authentication providers
- Explain basic LDAP concepts
- Configure a password validation provider
- Import/export security data

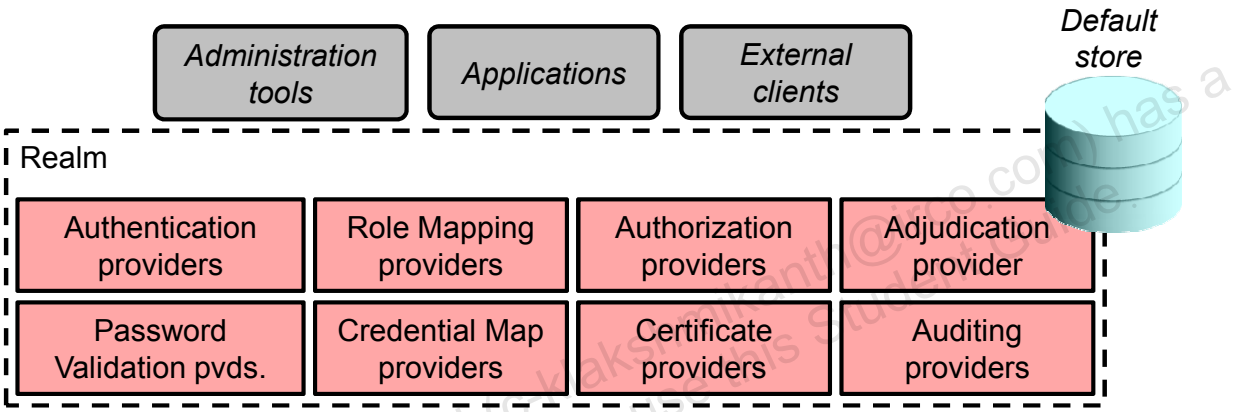
Road Map

- WebLogic Authentication
  - Security Realm Review
  - Custom Realms
  - Authentication Provider Review
  - Control Flags
- LDAP Providers
- Database Providers
- Password Validation
- Security Migration

Security Realm Review

A security realm:

- Handles security logic and decisions for a domain
- Consists of a series of pluggable providers
- Is scoped to all servers in a domain



Security Realm Review

The security service in WebLogic Server simplifies the configuration and management of security while offering robust capabilities for securing your WebLogic Server deployment. Security realms act as a scoping mechanism. Each security realm consists of a set of configured security providers, users, groups, security roles, and security policies. You can configure multiple security realms in a domain; however, only one can be the active security realm.

A security policy is an association between a WebLogic resource and one or more users, groups, or security roles. Security policies protect the WebLogic resource against unauthorized access. A WebLogic resource has no protection until you create a security policy for it.

A security provider store contains the users, groups, security roles, security policies, and credentials used by some types of security providers to provide their services. For example, an authentication provider requires information about users and groups; an Authorization provider requires information about security policies; a Role Mapping provider requires information about security roles, and a Credential Mapping provider requires information about credentials to be used to remote applications. These security providers need this information to be available in a database to function properly.

Security Realm Review (continued)

Determining what to do if multiple Authorization providers' access decisions do not agree on an answer is the function of an Adjudication provider. The Adjudication provider resolves authorization conflicts by weighing each access decision and returning a final result.

The Certificate Lookup and Validation providers complete certificate paths and validate X509 certificate chains. If multiple providers are configured, a certificate or certificate chain must pass validation with all them in order for the certificate or certificate chain to be accepted.

An Auditing provider collects, stores, and distributes information about operating requests and the outcome of those requests for the purposes of nonrepudiation. An Auditing provider makes the decision about whether to audit a particular event based on specific audit criteria, including audit severity levels.

Security Provider Stores

A persistent store is assigned to a security realm to persist assets such as:

- Users and groups
- Roles
- Policies
- Credential maps
- Certificates

Some providers use the default realm store while others use an external system.

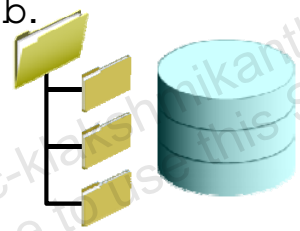
Security Provider Stores

The default Auditing and Adjudication providers do not use the persistent stores configured for their parent security realm.

If you have multiple security providers of the same type configured in the same security realm, these security providers may use the same security provider database. For example, if you configure two WebLogic authentication providers in the default security realm (called myrealm), both WebLogic authentication providers will use the same location in the embedded LDAP server as their security provider database and thus will use the same users and groups. Furthermore, if you or an administrator add a user or group to one of the WebLogic authentication providers, you will see that user or group appear for the other WebLogic authentication provider as well.

Store Implementations

- Two default options exist:
  - An embedded LDAP server, which runs on the Administration Server and is replicated to managed servers
  - A database
- To configure the database option, use either of the following:
  - The Configuration Wizard
  - The administration console or WLST
- The required schema files are located at `WL_HOME/server/lib`.



Store Implementations

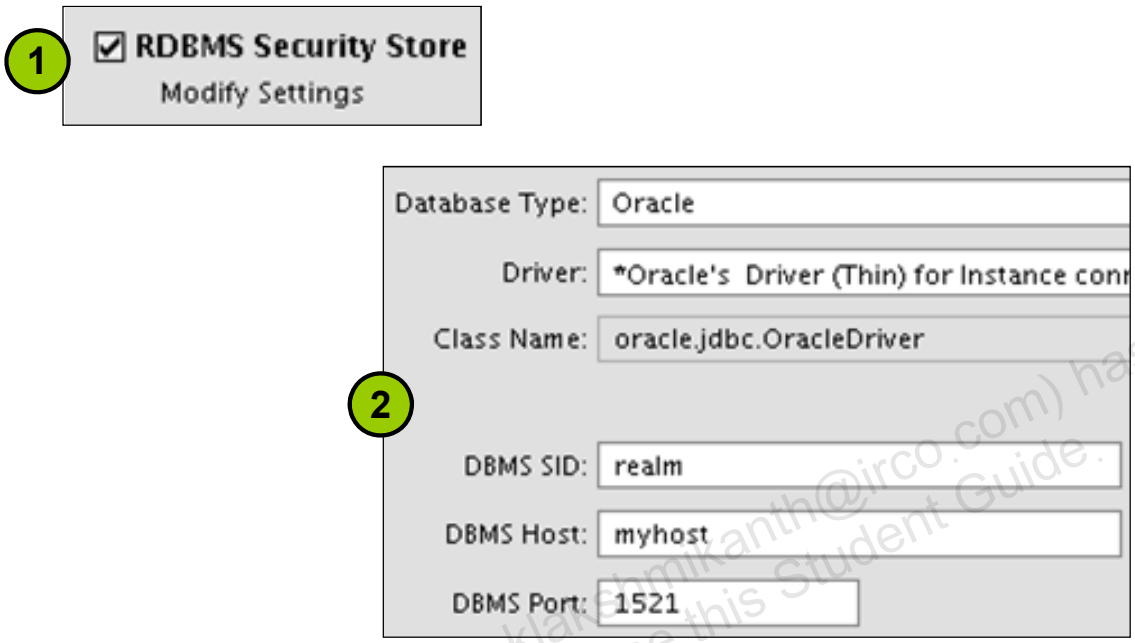
WebLogic Server uses its embedded LDAP server as the database that stores user, group, security roles, and security policies for the WebLogic security providers. The embedded LDAP server is a complete LDAP server that is production quality for reasonably small environments (10,000 or fewer users). For applications that need to scale above this recommendation, the embedded LDAP server can serve as an excellent development, integration, and testing environment for future export to an external LDAP server for production deployment.

When the RDBMS security store is configured in a security realm, any of the following security providers that has been created in the security realm automatically uses only the RDBMS security store, and not the embedded LDAP server:

- XACML Role Mapping and Authorization
- Default, PKI, and SAML Credential Mapping
- SAML Identity Assertion

Other security providers continue to use their default stores; for example, the WebLogic authentication provider continues to use the embedded LDAP server. Note that the use of the RDBMS security store is required to use SAML 2.0 services in two or more WebLogic Server instances in a domain, such as in a cluster.

Configuring the RDBMS Store by Using the Configuration Wizard



Configuring the RDBMS Store by Using the Configuration Wizard

To use the RDBMS security store, the preferred approach is first to create a domain in which the external RDBMS server is configured. Before booting the domain, you create the tables that are required by the RDBMS security store. The WebLogic Server installation directory contains a set of SQL scripts that create these tables for each supported database.

1. The Configuration Wizard includes an option to configure the RDBMS Security Store. This option is available from the Customize Environment and Services Settings page.
2. Select the specific database system that you want to use as the RDBMS security store, configure the database connection settings, and then test the connection settings.

The databases that appear in the Database Type list are not exclusive to only those supported for the RDBMS security store. Make sure that you choose a database type that is supported for this purpose, which includes Oracle (9*i*,10*g*,11*g*) and DB2 (9.2, 9.5).

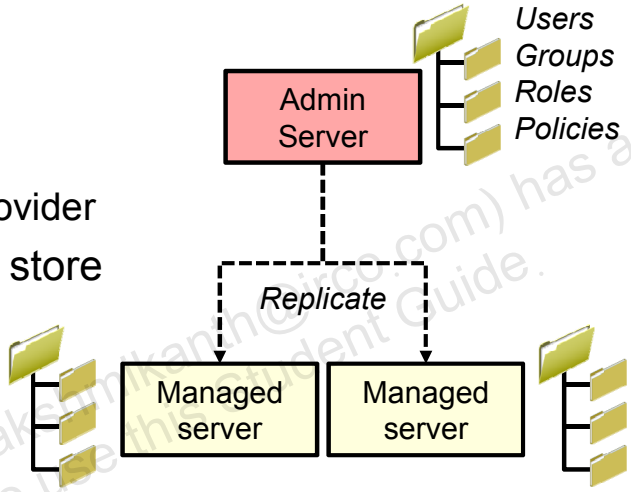
To create a new RDBMS security store using WLST, use the RDBMSSecurityStoreMBean:

```
rdbms = realm.create("myRDBMSSecurityStore", "RDBMSSecurityStore")
rdbms.setUsername('ortiz')
rdbms.setPassword('weblogic')
rdbms.setConnectionURL('jdbc:bea:oracle://avitek21:1521')
```

Default Security Configuration

A new domain includes a default realm that:

- Includes some basic providers such as:
  - Default authenticator
  - Default identity asserter
  - XACML role mapper
  - XACML authorizer
  - Default adjudicator
  - Default certificate path provider
- Uses the embedded LDAP store



Default Security Configuration

To simplify the configuration and management of security, WebLogic Server provides a default security configuration. In the default security configuration, myrealm is set as the default (active) security realm, and the WebLogic Adjudication, Authentication, Identity Assertion, Credential Mapping, CertPath, EXtensible Access Control Markup Language (XACML) Authorization and XACML Role Mapping providers are defined as the security providers in the security realm.

The WebLogic Server embedded LDAP server for a domain consists of a master LDAP server, maintained in the domain's Administration Server, and a replicated LDAP server maintained in each managed server in the domain. When changes are made using a managed server, updates are sent to the embedded LDAP server on the Administration Server. The embedded LDAP server on the Administration Server maintains a log of all changes. The embedded LDAP server on the Administration Server also maintains a list of managed servers and the current change status for each one. The embedded LDAP server on the Administration Server sends appropriate changes to each managed server and updates the change status for each server. This process occurs when an update is made to the embedded LDAP server on the Administration Server. However, depending on the number of updates, it may take several seconds or more for the change to be replicated to the managed server.

Security Customization Approaches

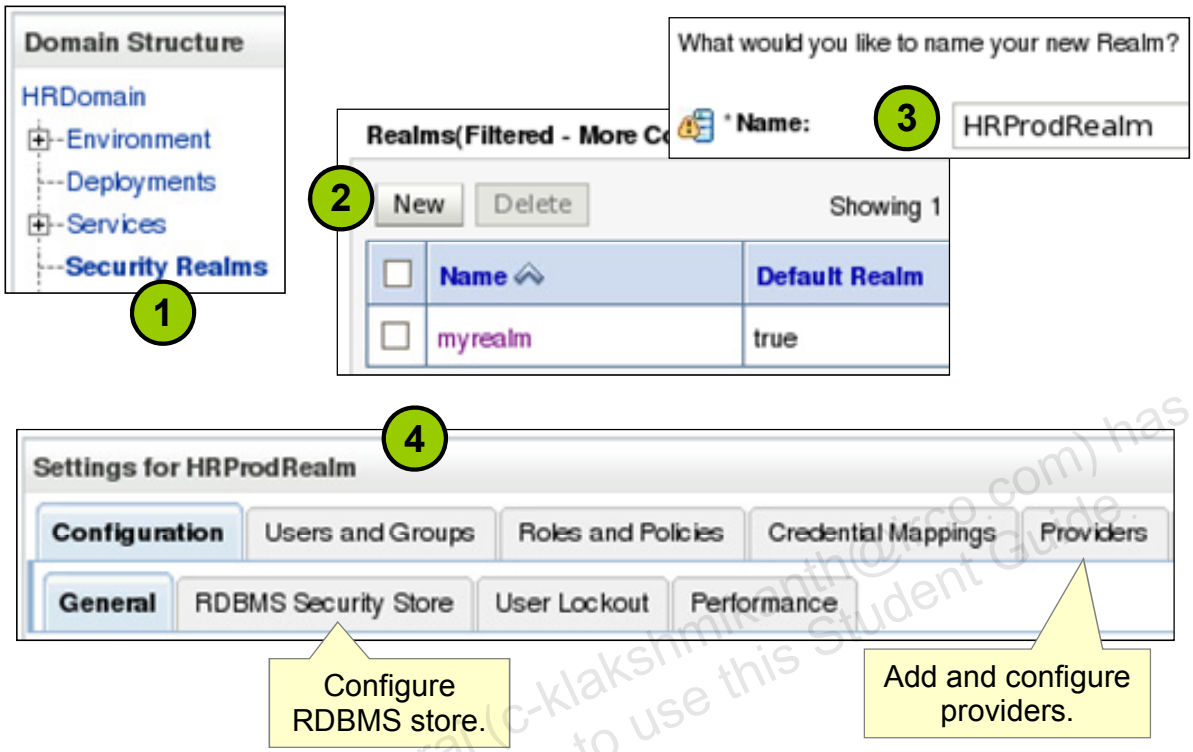
- Create an entire new security realm and add required providers:
  - Only one realm can be active at a time.
  - Quickly validate multiple security configurations.
- Add, remove, and configure providers in the default realm.
- Develop custom providers and add them to a realm.

Security Customization Approaches

The easiest way to customize the default security configuration is to add the security providers you want to the default security realm (myrealm). However, Oracle recommends instead that you customize the default security configuration by creating an entirely new security realm. This preserves your ability to revert more easily to the default security configuration. You configure security providers for the new realm, migrate any security data, such as users as groups, from the existing default realm, and then set the new security realm as the default realm.

Configure the required security providers for the security realm. A valid security realm requires an authentication provider, an Authorization provider, an Adjudication provider, a Credential Mapping provider, a Role Mapping provider, and a CertPathBuilder. Optionally, define Identity Assertion, Auditing, and Certificate Registry providers. If you configured the Default Authentication, Authorization, Credential Mapping or Role Mapping provider or the Certificate Registry in the new security realm, verify that the settings of the embedded LDAP server are appropriate.

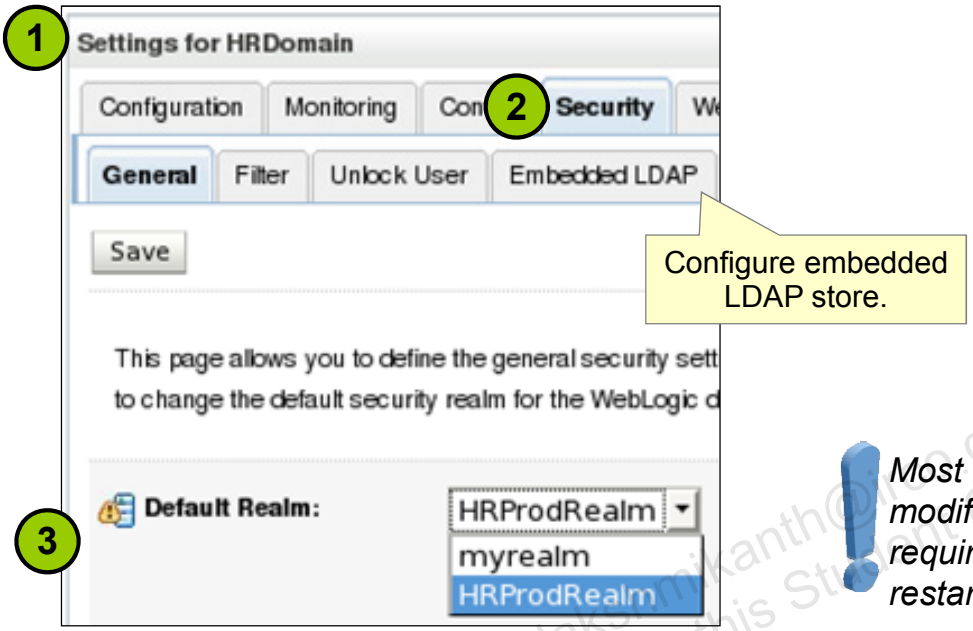
Creating a New Security Realm



Creating a New Security Realm

1. In the left pane, select Security Realms. All the security realms available for the WebLogic domain are listed in the Realms table.
2. Click New.
3. Enter the name of the new security realm. Click OK.
4. Click the new realm. Use the Providers tab to add any security providers to the realm. Optionally, use the Configuration > RDBMS Security Store tab to use this store type as the realm default instead of the embedded LDAP.
5. Reboot your domain. If you do not reboot WebLogic Server, you cannot set the realm to the default security realm.

Activating a Security Realm



Activating a Security Realm

1. In the left pane, select your domain name.
2. Click the Security tab.
3. In the Default Realm field, select the security realm that this domain should use.
4. Reboot your domain.

You can also use the domain's Security > Embedded LDAP tab to customize the default security and cache settings, if desired. The default behavior of the embedded LDAP server is to allow access only from the Admin account in WebLogic Server. The WebLogic security providers use only the Admin account to access the embedded LDAP server. If you are not planning to access the embedded LDAP server from an external LDAP browser or if you are planning only to use the Admin account, you do not need to customize these settings.

Security Realm WLST Example

Create and activate a new security realm:

```
edit()
startEdit()

security = getMBean('/SecurityConfiguration/HRDomain')
realm = security.createRealm('HRProdRealm')
realm.createAuthenticationProvider('DefaultAuthenticator',
   'weblogic.security.providers.authentication.
   DefaultAuthenticator')

... create remaining providers ...

security.setDefaultRealm(realm)

save()
activate(block='true')
```


Security Realm WLST Example

Refer to the following MBeans in the documentation:

- `SecurityConfigurationMBean`
- `RealmMBean`
- `AuthenticationProviderMBean`

Authentication Provider Review

Authentication providers are organized into two categories:

- *Authenticators:*
  - Establish the user's identity given some credentials (password and so on)
  - Can associate multiple identities or "principals" with a single user, such as groups
- *Identity asserters:*
  - Validate tokens claiming a user has already been authenticated
  - Allow WebLogic Server to participate in single sign-on (SSO) solutions
  - Can map the token to a local user and use authenticators to look up principals

Authentication Provider Review

Authentication providers are used to prove the identity of users or system processes. Authentication providers also remember, transport, and make that identity information available to various components of a system (via subjects) when needed.

Both users and groups can be used as principals by application servers like WebLogic Server. A principal is an identity assigned to a user or group as a result of authentication. The Java Authentication and Authorization Service (JAAS) requires that subjects be used as containers for authentication information, including principals. Each principal stored in the same subject represents a separate aspect of the same user's identity, much like cards in a person's wallet.

An *identity assertion provider* is a specific form of authentication provider that allows users or system processes to assert their identity using tokens supplied by clients. Typical token types include X509 certificates, SAML, and Kerberos. Identity assertion providers enable perimeter authentication and support SSO. For example, an identity assertion provider can generate a token from a digital certificate, and that token can be passed around the system so that users are not asked to sign on more than once.

Unlike in a simple authentication situation, identity assertion providers do not verify credentials such as usernames and passwords. They verify that the user exists.

Available Authentication Providers

- Available authenticators include:
  - Default
  - LDAP (generic and vendor-specific)
  - Database
  - Windows NT
  - SAML
- Available identity asserters include:
  - Default
  - LDAP X509
  - SAML
  - Negotiate (SPNEGO)

Available Authentication Providers

The default WebLogic Server authentication provider manages users and groups in the embedded LDAP server.

LDAP authentication providers access external LDAP stores. WebLogic Server provides LDAP authentication providers that access Oracle Internet Directory, Oracle Virtual Directory, Open LDAP, Netscape iPlanet, Microsoft Active Directory and Novell NDS stores.

An RDBMS authentication provider is a username- and password-based authentication provider that uses a relational database (rather than an LDAP directory) as its data store for user, password, and group information.

The SAML authentication provider may be used with the SAML 1.1 or SAML 2.0 identity assertion provider to allow virtual users to log in via SAML. This provider creates an authenticated subject using the user name and groups retrieved from a SAML assertion.

The Windows NT authentication provider uses account information defined for a Windows NT domain to authenticate users and groups and to permit Windows NT users and groups to be listed in the administration console.

Available Authentication Providers (continued)

The default identity assertion provider supports identity assertion with X509 certificates and CORBA Common Secure Interoperability version 2 (CSI v2). You can also map the tokens authenticated by the identity assertion provider to users in the security realm.

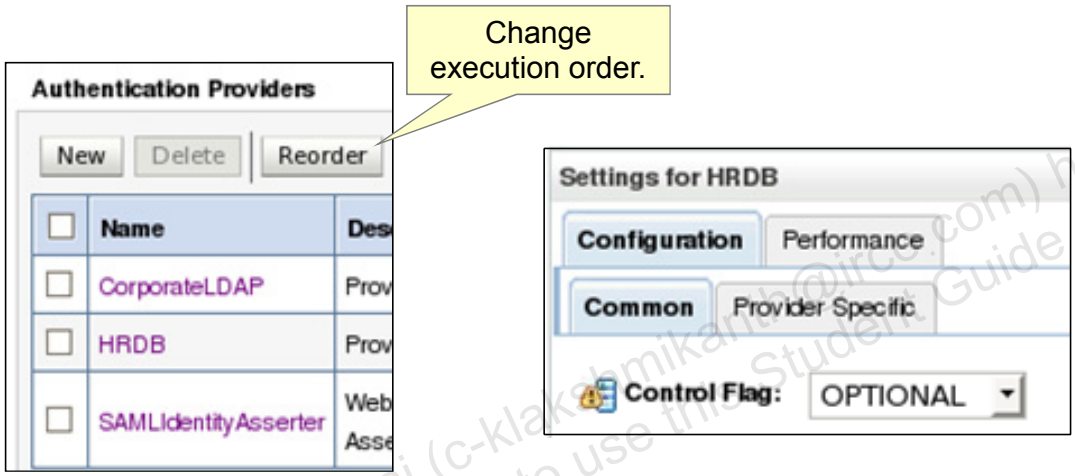
The LDAP X509 identity assertion provider receives an X509 certificate, looks up the LDAP object for the user associated with that certificate, ensures that the certificate in the LDAP object matches the presented certificate, and then retrieves the name of the user from the LDAP object.

The SAML identity assertion provider acts as a consumer of SAML security assertions, allowing WebLogic Server to participate in SSO solutions for Web or Web service applications. It validates assertions by checking the signature and validating the certificate for trust based on data configured for the associated partner. The provider then extracts the identity information contained in the assertion, and maps it to a local subject in the security realm.

The Negotiate identity assertion provider enables SSO with Microsoft clients. The provider decodes simple and protected Negotiate (SPNEGO) tokens to obtain Kerberos tokens, validates the Kerberos tokens, and maps Kerberos tokens to WebLogic users.

Multiple Authentication Providers

- A single realm can support multiple authentication providers.
- For authenticators, *control flags* determine the processing logic as each provider is executed.



Multiple Authentication Providers

Each security realm must have at least one authentication provider configured. The WebLogic Security Framework supports multiple authentication providers for multipart authentication. Therefore, you can use multiple authentication providers as well as multiple types of authentication providers in a security realm.

The order in which WebLogic Server calls multiple authentication providers can affect the overall outcome of the authentication process. The Authentication Providers table lists the authentication providers in the order in which they will be called. By default, authentication providers are called in the order in which they were configured. Use the Reorder button on the Security Realms > Providers > Authentication page in the administration console to change the order in which authentication providers are called by WebLogic Server and listed in the console.

When you configure multiple authentication providers, also use the Control Flag for each provider to control how the authentication providers are used in the login sequence. When additional authentication providers are added to an existing security realm, by default the Control Flag is set to OPTIONAL. If necessary, change the setting of the Control Flag and the order of authentication providers so that each authentication provider works properly in the authentication sequence.

Control Flags

Flag	Success Action	Failure Action
REQUIRED	Execute next provider	Execute next provider, but outcome still failure
REQUISITE	Execute next provider	Execute only REQUIRED providers, but outcome still failure
SUFFICIENT	Execute only REQUIRED providers	Execute next provider
OPTIONAL	Execute next provider	Execute next provider

Control Flags

All REQUIRED modules must be invoked, and each must successfully validate the user.

Any REQUISITE module that gets invoked must successfully validate the user.

If a SUFFICIENT module successfully validates the user, the overall success depends on the success of all REQUIRED modules, and any REQUISITE modules invoked before the SUFFICIENT module.

If the login sequence consists only of OPTIONAL modules, at least one module must successfully validate the user.

Administration Groups

At least one authentication provider must exist that associates users with groups that have administrative rights.

Group	Default Capability (via roles and policies)
Administrators	Full administrative access to the domain and its applications
Operators	View domain configuration, start, and stop servers
Deployers	View domain configuration, deploy, and update applications
Monitors	View domain configuration
AppTesters	Access applications running in admin mode through a dedicated admin network channel

Administration Groups

For efficient security management, Oracle recommends adding users to groups. A group is a collection of users who usually have something in common, such as working in the same department in a company. By default, the Role Mapping and Authorization providers include policies that grant specific groups different administrative rights in your domain. You can change these policies and roles if desired, but be careful that you do not accidentally make your domain inaccessible. Also, having at least two administrators at all times helps protect against a single user being locked out from a potential security breach.

The Administrators group contains your domain administrative account and is assigned to the Admin role. Similarly, the OracleSystemGroup group contains a user named OracleSystemUser and is assigned to the OracleSystemRole role.

The remaining administrative groups have no users by default, but are mapped to these roles:

- **Deployers group:** Deployer role
- **Operators group:** Operator role
- **Monitors group:** Monitor role
- **AppTesters group:** AppTester role
- **CrossDomainConnectors group:** CrossDomainConnector role
- **AdminChannelUsers group:** AdminChannelUser role

Section Summary

In this section, you should have learned how to:

- Describe the WebLogic Server security realm architecture
- Create a new security realm
- Compare authenticators and identity assertors
- Use control flags to combine multiple providers

Road Map

- WebLogic Authentication
- LDAP Providers
  - LDAP Concepts
  - System Groups
  - Dynamic Groups
  - Failover and Caching
- Database Providers
- Password Validation
- Security Migration

Lightweight Directory Access Protocol (LDAP)

- LDAP:
  - Is a TCP/IP protocol
  - Provides a hierarchical lookup and search service
  - Models information as a tree of entries, whose attributes are defined by a schema or “object class”
  - Defines default schemas for common entities like people and groups
  - Supports SSL
- Entries:
  - Identify their locations in the tree using a distinguished name (DN)
  - Can be referrals that link to other LDAP servers

Lightweight Directory Access Protocol (LDAP)

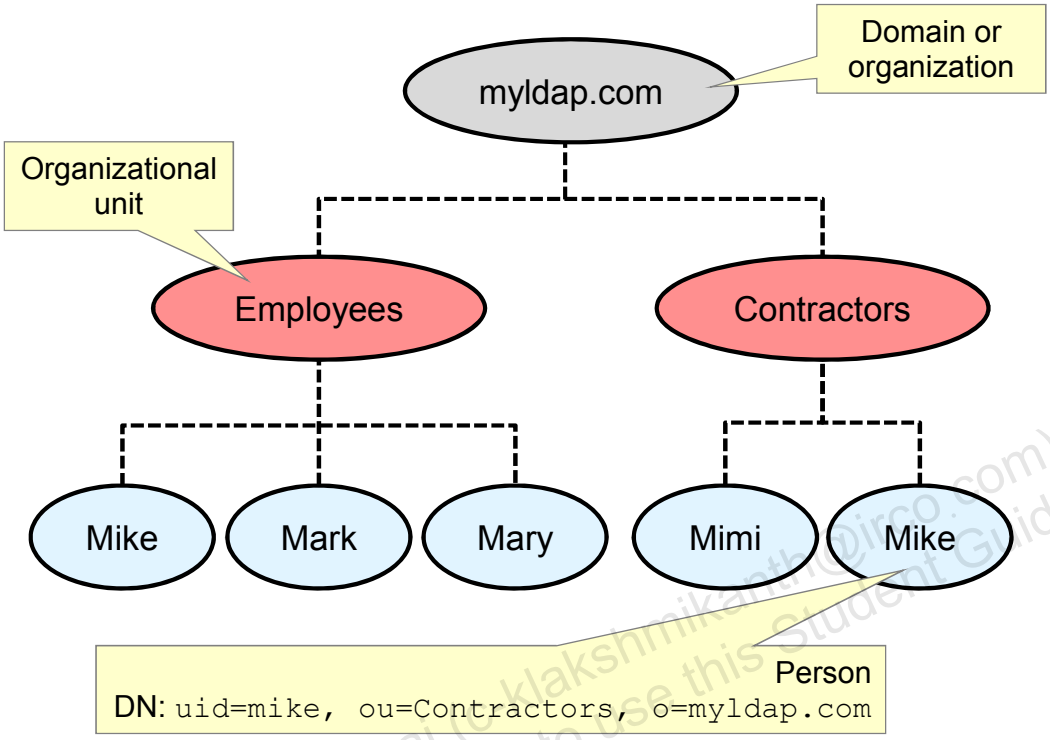
The Lightweight Directory Access Protocol, better known as LDAP, is based on the X.500 standard, but is significantly simpler and more readily adapted to meet custom needs. Unlike X.500, LDAP supports TCP/IP, which is necessary for Internet access. The core LDAP specifications are all defined in Request for Comments (RFCs).

LDAP is a protocol that provides access to a compliant directory via TCP/IP. The strengths of LDAP-compliant directories include speed, simplicity, and the ability to be replicated and distributed across several servers. An LDAP directory can be used to store a great deal of information from user login credentials to company telephone directories.

Unlike databases that are designed for processing hundreds or thousands of changes per minute, LDAP directories are heavily optimized for read performance. LDAP is intentionally designed for environments where search operations are much more common than modify operations.

LDAP Version 3 implements a referral mechanism that allows servers to return references to other servers as a result of a directory query. This makes it possible to distribute directories globally by partitioning a directory information tree (DIT) across multiple LDAP servers.

LDAP Structure



LDAP Structure

Directories are viewed as a tree, like a computer's file system. Each entry in a directory is called an object. These objects are of two types: containers and leaves. A container is like a folder; it contains other containers or leaves. A leaf is simply an object at the end of a tree. A tree cannot contain any arbitrary set of containers and leaves. It must match the schema defined for the directory.

The top level of the LDAP directory tree is the base, referred to as the base DN. A base DN can be one of several forms. Here are some examples:

- A domain name, broken into components (dc=Acme,dc=com)
- An organization name (o=Acme Corp)
- An organization name along with a country (o=Acme Corp,c=India)

Organizational units are standard LDAP object classes that act as containers for other entries. The identifying attribute for an organizational unit is "ou." The standard LDAP schema also defines a person class and a group class, which is a collection of people.

The person type also includes other attributes such as Common Name (cn), Unique Identifier (uid), Last Name (sn), and Password (userpassword).

LDAP Search Operations

Searching for LDAP entries involves:

1. The base DN from which to start searching
2. A search filter that specifies the:
   - Search criteria in terms of attribute values
   - The type or “object class” of the desired entries
3. An indication whether or not the search should include any child entities

An LDAP search filter that finds all people whose user ID begins with “m,” while ignoring those whose name is “Mike”:

```
(&(uid=m*)(!cn=Mike*)((objectclass=person))
```


LDAP Search Operations

The “&” represents a logical “and” when combining multiple expressions that have been grouped together in parentheses. Similarly, the “|” represents a logical “or,” and a “!” represents a logical “not.”

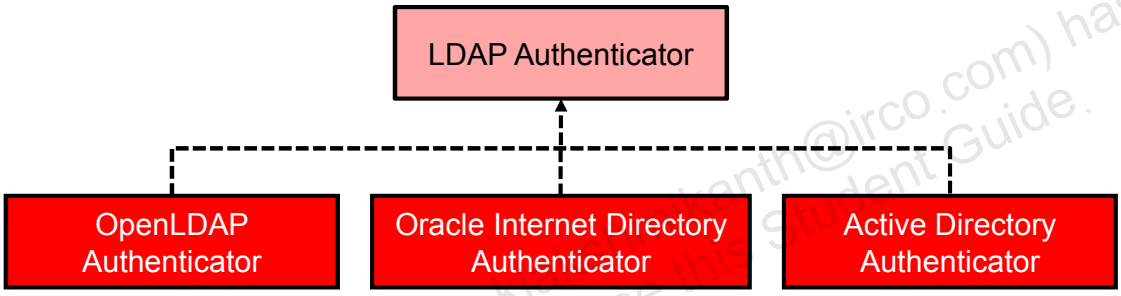
Search filters can specify one or more object classes. Here is an example:

```
(&(&(objectClass=person)(objectClass=organizationalPerson))(objectC
lass=user))
```

LDAP Authentication Providers

WebLogic Server includes:

- A base LDAP authenticator that can be configured to support any compliant vendor
- Vendor-specific LDAP authenticators, whose attributes are set to vendor-specific defaults for convenience



LDAP Authentication Providers

Each LDAP authentication provider stores user and group information in an external LDAP server. The providers differ primarily in how they are configured by default to match typical directory schemas for their corresponding LDAP server. For example, the generic authenticator is configured to use a person's common name (cn) as a user ID, while by default Oracle Internet Directory uses the “uid” attribute for this purpose. Similarly, the names of object classes used to represent people or groups may vary from vendor to vendor. For example, the generic authenticator is configured to use the object class “groupofuniquenames,” while by default Oracle Internet Directory uses the object class “groupofnames.”

WebLogic Server does not support or certify any particular LDAP servers. Any LDAP v2 or v3 compliant LDAP server should work with WebLogic Server.

If an LDAP authentication provider is the only configured authentication provider for a security realm, you must have the Admin role to boot WebLogic Server and use a user or group in the LDAP directory. If the LDAP user who boots WebLogic Server is not properly added to a group that is assigned to the Admin role, and the LDAP authentication provider is the only authentication provider with which the security realm is configured, WebLogic Server cannot be booted.

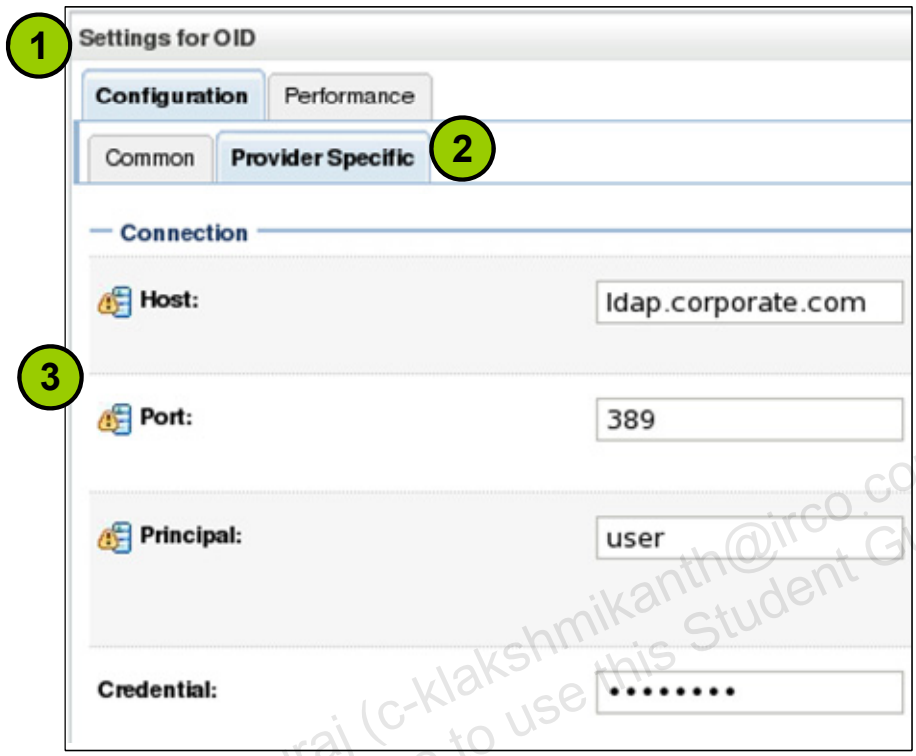
Available LDAP Authentication Providers

- The available LDAP authentication providers include:
  - LDAP Authenticator
  - Oracle Internet Directory Authenticator
  - Oracle Virtual Directory Authenticator
  - IPlanet Authenticator
  - Active Directory Authenticator
  - Novell Authenticator
  - OpenLDAP Authenticator
- These providers:
  - Can be used to change passwords of existing users
  - Cannot be used to create, update, or delete users and groups

Available LDAP Authentication Providers

The LDAP authentication providers in this release of WebLogic Server are configured to work readily with the Oracle Internet Directory, Oracle Virtual Directory, SunONE (iPlanet), Active Directory, Open LDAP, and Novell NDS LDAP servers. You can use an LDAP authentication provider to access other types of LDAP servers. Choose either the generic LDAP authenticator or an existing LDAP provider that most closely matches the new LDAP server and customize the existing configuration to match the directory schema and other attributes for your LDAP server.

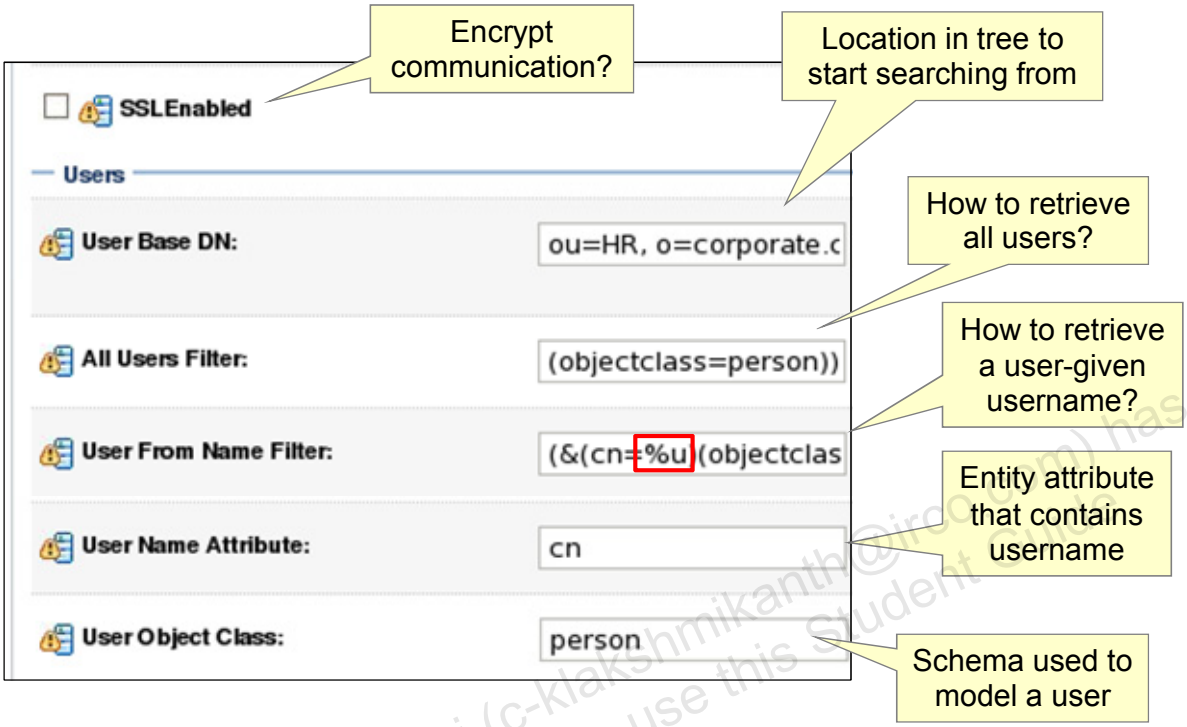
Configuring an LDAP Provider: Connection



Configuring an LDAP Provider: Connection

1. After adding an LDAP authenticator to a realm's list of authentication providers, edit it.
2. Click Configuration > Provider Specific.
3. Enter values for the following connection attributes:
   - **Host:** A single hostname or IP address, or a comma-separated list of hostnames and ports to try
   - **Port:** The host name or IP address of the LDAP server
   - **Principal:** The Distinguished Name (DN) of the LDAP user that WebLogic Server should use to connect to the LDAP server
   - **Credential:** The credential (usually a password) used to connect to the LDAP server
   - **SSL Enabled:** Specifies whether the SSL protocol should be used when connecting to the LDAP server. For a more secure deployment, Oracle recommends using the SSL protocol to protect communications between the LDAP server and WebLogic Server.

Configuring an LDAP Provider: Users

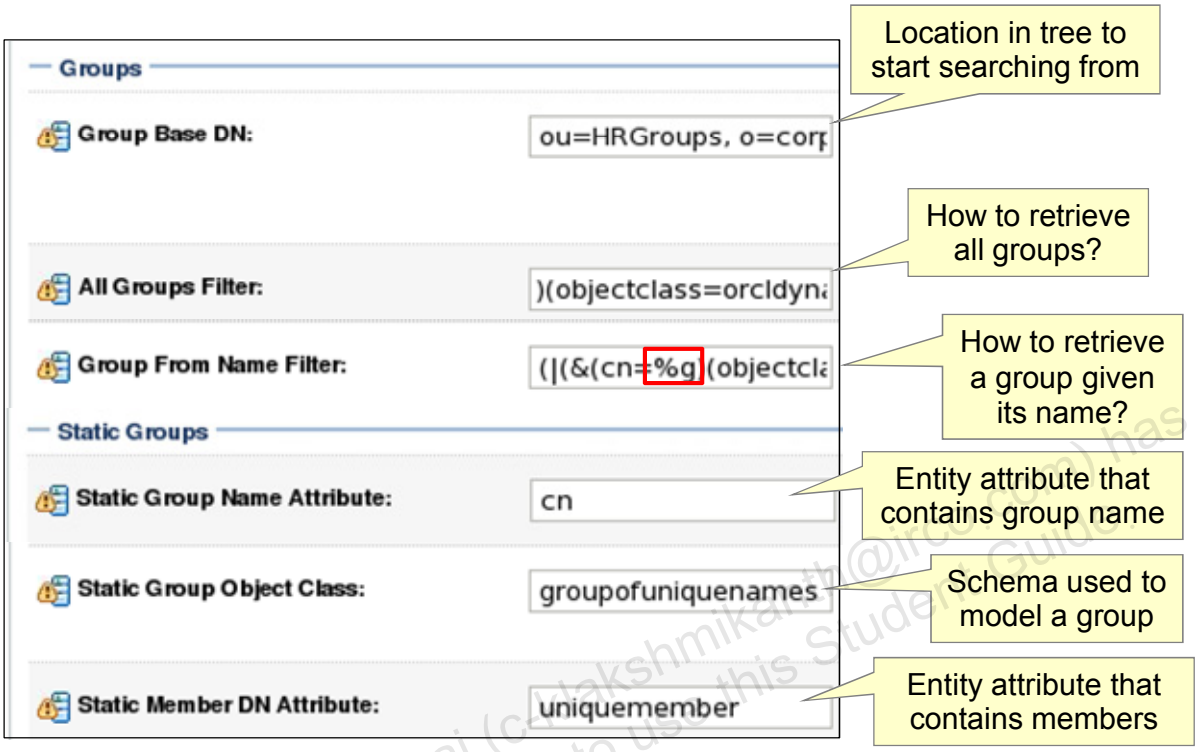


Configuring an LDAP Provider: Users

Enter values for any of the following user search attributes:

- **User Base DN:** The base distinguished name (DN) of the tree in the LDAP directory that contains users
- **All Users Filter:** The LDAP filter expression used to retrieve all users. If not specified, a simple default filter is generated based on the user object class.
- **User From Name Filter:** The LDAP filter expressions used to locate a user entry given its user ID. Use the token “%u” to indicate where the provider should insert the user ID before executing the search.
- **User Search Scope:** Specifies how deep in the LDAP directory tree the provider should search for users. Valid values are “subtree” and “onelevel.”
- **User Name Attribute:** The attribute of an LDAP user object that specifies the user’s login ID
- **User Object Class:** The LDAP object class that stores users
- **Use Retrieved User Name as Principal:** Specifies whether or not the username retrieved from the LDAP server should be used as the principal

Configuring an LDAP Provider: Groups



Configuring an LDAP Provider: Groups

Enter values for any of the following group search attributes:

- **Group Base DN:** The base distinguished name (DN) of the tree in the LDAP directory that contains group definitions
- **All Groups Filter:** An LDAP filter expression for finding all groups beneath the base group distinguished name (DN). If not specified, a simple default search filter is created based on the group object class.
- **Group From Name Filter:** An LDAP filter expression for finding a group given the name of the group. If not specified, a simple default search filter is created based on the group schema.
- **Group Search Scope:** Specifies how deep in the LDAP directory tree to search for groups. Valid values are “subtree” and “onelevel.”
- **Static Group Name Attribute:** The attribute of a group object that specifies the name of the group
- **Static Group Object Class:** The name of the LDAP object class that stores groups
- **Static Member DN Attribute:** The attribute of a group object that specifies the distinguished names (DNs) of the members of the group
- **Static Group DNs from Member DN Filter:** Given the DN of a member of a group, returns the DNs of the groups that contain that member

Configuring an LDAP Provider: Subgroups

- Group members may include other groups.
- To improve performance, you can limit the depth that the provider will search for subgroups.

ORACLE

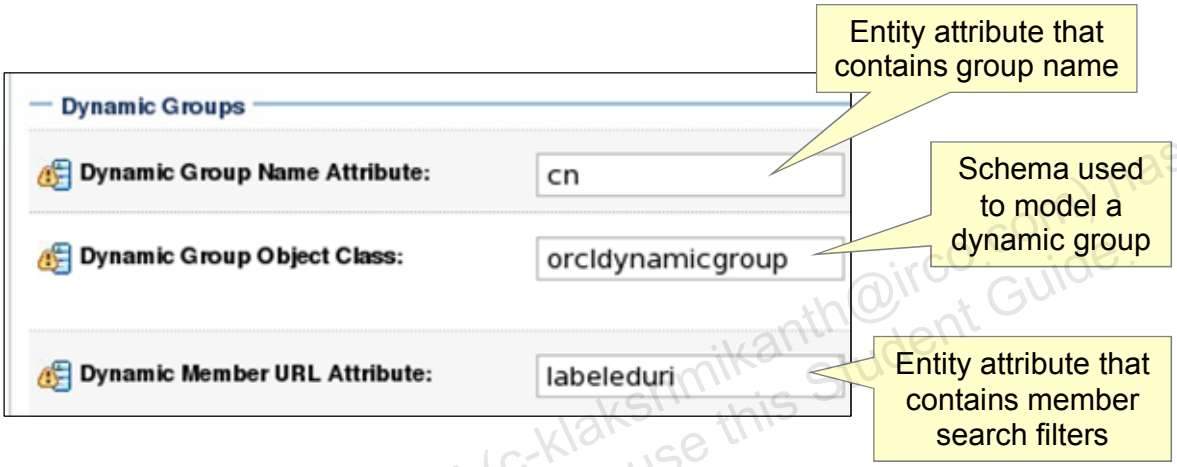
Configuring an LDAP Provider: Subgroups

Enter values for any of the following attributes that apply to subgroup searching:

- **Group Membership Searching:** Specifies whether recursive group searches into nested groups are unlimited or limited. For configurations that use only the first level of nested group hierarchy, this attribute allows improved performance during user searches by limiting the search to the first level of the group.
- **Max Group Membership Search Level:** Specifies how many levels of group membership can be searched. This setting is valid only if Group Membership Searching is set to “limited.” A value of 0 indicates that only direct groups will be found. That is, when searching for membership in Group A, only direct members of Group A will be found. If Group B is a member of Group A, the members will not be found by this search. Any non-zero value indicates the number of levels to search. For example, if this attribute is set to 1, a search for membership in Group A will return direct members of Group A. If Group B is a member of Group A, the members of Group B will also be found by this search. However, if Group C is a member of Group B, the members of Group C will not be found by this search.

Configuring an LDAP Provider: Dynamic Groups

- Instead of a list of users, dynamic groups contain a list of search filters, each of which returns zero or more users.
- Member search filters are expressed as URLs.



Configuring an LDAP Provider: Dynamic Groups

Many LDAP servers have a concept of dynamic groups or virtual groups. These are groups that, rather than consisting of a list of users and groups, contain some policy statements, queries, or code that define the set of users that belong to the group. Even if a group is marked dynamic, users must log out and log back in before any changes in their group memberships take effect. The term “dynamic” describes the means of defining the group and not any runtime semantics of the group within WebLogic Server.

The provider attributes for dynamic groups are very similar to static ones, but the following additional attributes are also available:

- **Dynamic Member URL Attribute:** The attribute of the dynamic LDAP group object that specifies the URLs of the members of the dynamic group
- **User Dynamic Group DN Attribute:** A user attribute indicating its dynamic group membership. If such an attribute does not exist, the provider determines if a user is a member of a group by evaluating the URLs on the dynamic group. If a group contains other groups, WebLogic Server evaluates the URLs on any of the descendants (subgroups) as well.

LDAP Provider WLST Example

Add an LDAP authenticator to a realm:

```
edit()
startEdit()

realm = getMBean('/SecurityConfiguration/HRDomain/Realms/
   HRProdRealm')
provider = realm.createAuthenticationProvider('OID',
   'weblogic.security.providers.authentication.
   OracleInternetDirectoryAuthenticator')
provider.setControlFlag('REQUIRED')
provider.setHost('ldap.corporate.com')
provider.setPort(389)
provider.setPrincipal('user')
provider.setCredential('password')
provider.setUserBaseDN('ou=HRUsers,o=corporate.com')
...
save()
activate(block='true')
```

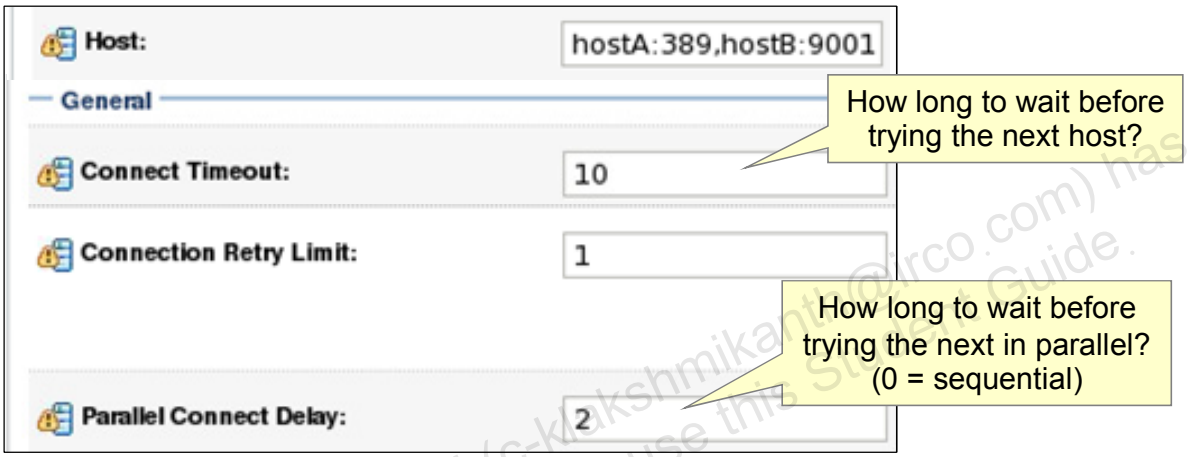

LDAP Provider WLST Example

Refer to the following MBeans in the documentation:

- `SecurityConfigurationMBean`
- `RealmMBean`
- `AuthenticationProviderMBean`
- `LDAPAuthenticatorMBean (a subclass of AuthenticationProviderMBean)`
- `OracleInternetDirectoryAuthenticatorMBean (a subclass of LDAPAuthenticatorMBean)`

LDAP Failover

- The **Host** attribute supports a list of candidate servers for high availability.
- Connection attempts can be made sequentially or in parallel.



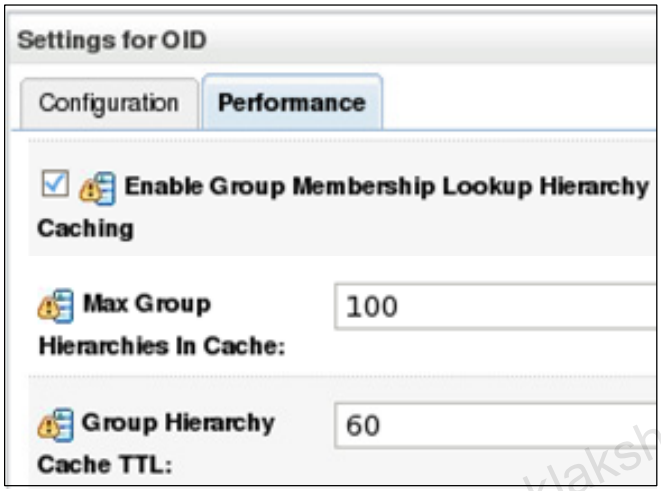
LDAP Failover

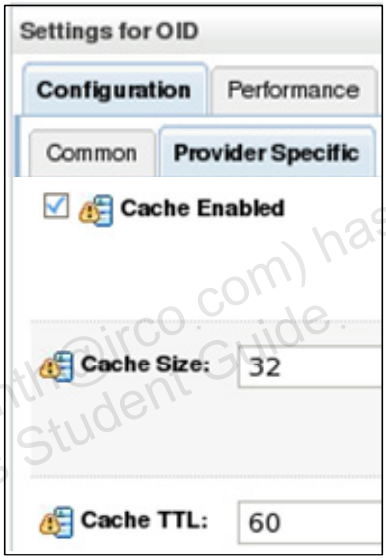
You can configure an LDAP provider to work with multiple LDAP servers and enable failover if one LDAP server is not available. Use the Host attribute to specify the names of the additional LDAP servers. Each host name may include a trailing comma and a port number. Also configure the following additional attributes:

- **Connect Timeout:** Specifies the maximum number of seconds to wait for the connection to the LDAP server to be established. If set to 0, there is no maximum time limit and WebLogic Server waits until the TCP/IP layer times out to return a connection failure.
- **Parallel Connect Delay:** Specifies the number of seconds to delay when making concurrent attempts to connect to multiple servers. An attempt is made to connect to the first server in the list. The next entry in the list is tried only if the attempt to connect to the current host fails. This setting might cause your application to block for an unacceptably long time if a host is down. If the value is greater than 0, another connection setup thread is started after the specified number of delay seconds has passed. If the value is 0, connection attempts are serialized.

LDAP Caching

- All authenticators can cache a group's member list.
- LDAP Authenticators can also cache individual entries.





LDAP Caching

Enabling a cache involves a trade-off of performance and accuracy. Using a cache means that data is retrieved faster, but runs the risk that the data may not be the latest available.

The time-to-live (TTL) setting specifies how long you are willing to accept potentially stale data. This depends a lot on your particular business needs. If you frequently change group memberships for users, then a long TTL could mean that group-related changes will not show up for a while, and you may want a short TTL. If group memberships almost never change after a user is added, a longer TTL may be fine. TTL attributes are specified in seconds.

The cache size is related to the amount of memory you have available, as well as the cache TTL. Consider the number of entries that might be loaded in the span of the TTL, and size the cache in relation to that number. A longer TTL will tend to require a larger cache size. For group membership caching, specify the number of groups to cache. For basic entry caching, specify the maximum size of the cache in kilobytes.

LDAP X509 Identity Asserter Overview

- The default identity asserter can map the subject found in a supplied X509 certificate to a local principal (user or group).
- The LDAP X509 identity asserter:
  - Checks incoming certificates against those registered in an external LDAP server
  - Obtains the user associated with the certificate from the LDAP server and checks that it matches the incoming subject
  - Requires that an LDAP Authenticator be configured to determine user's group membership

LDAP X509 Identity Asserter Overview

Identity assertion providers validate specific tokens found in client requests. Unlike authenticators, identity asserters must trust that the user has submitted sufficient proof to log in and not require a password or some other proof material. They can also assign principals (user or group names) to the incoming request based on the token data.

Identity assertion providers can support more than one token type, but only one token type per provider can be active at a given time. The default WebLogic identity asserter can validate X509 certificates provided by clients.

The LDAP X509 identity assertion provider receives an X509 certificate, looks up the LDAP object for the user associated with that certificate, ensures that the certificate in the LDAP object matches the presented certificate, and then retrieves the name of the user from the LDAP object. A correlation must exist between the Subject DN in the certificate and the location of the object for that user in the LDAP server. The LDAP object for the user must also include configuration information for the certificate and the username that will be used in the Subject.

LDAP X509 Identity Asserter Overview (continued)

Configure the LDAP X509 identity assertion provider to search the LDAP server to locate the LDAP object containing the username for a given certificate. This requires the following attributes:

- A base LDAP DN from which to start searching for objects that contain certificates and a search filter to limit which objects are retrieved. Include the source certificate's Subject DN in the filter.
- The object attribute that contains the certificate
- The object attribute that contains the username associated with the certificate

Section Summary

In this section, you should have learned how to:

- Explain how LDAP organizes information
- Create simple LDAP search filters
- List the available LDAP authenticators and identity asserters
- Configure user and group search settings for an LDAP authenticator
- Configure LDAP authenticator failover and caching

Practice 16-1: Authenticate Using an External LDAP

This practice covers the following topics:

- Configuring an external LDAP repository (OpenDS) with default WebLogic groups
- Adding an LDAP authentication provider to a security realm
- Using authentication provider control flags

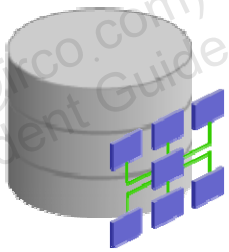
Road Map

- WebLogic Authentication
- LDAP Providers
- Database Providers
  - SQL Authenticator
  - Default Schema
- Password Validation
- Security Migration

Database Authentication Providers

All database authenticators share a common base implementation that supports:

- A JDBC data source
- Plain text passwords
- Passwords hashed using any of the algorithms supported by the Java Cryptography Extension (JCE)
- Nested groups
- Membership caching



Database Authentication Providers

In WebLogic Server, an RDBMS authentication provider is a username- and password-based authentication provider that uses a relational database (rather than an LDAP directory) as its data store for user, password, and group information.

The Java Cryptography Extension (JCE) encompasses the parts of the Java security API related to cryptography. It includes a “provider” architecture that allows for multiple and interoperable cryptography implementations. Providers refer to a package or set of packages that implement one or more cryptographic services, such as digital signature algorithms, message digest algorithms, and key conversion services. The default provide package includes an implementation of the MD5 (RFC 1321) and SHA-1 (NIST FIPS 180-1) message digest algorithms along with an implementation of the proprietary “SHA1PRNG” pseudo-random number generation algorithm, following the recommendations in the IEEE P1363 standard (Appendix G.7).

You can improve the performance of RDBMS authentication providers by caching the results of group hierarchy lookups. Use of this cache can reduce the frequency with which the RDBMS authentication provider needs to access the database. In the administration console, you can use the Performance page for your provider to configure the use, size, and duration of this cache.

Available Database Authenticators

- The *SQL Authenticator*:
  - Lets you configure individual SQL statements to search for and update users and groups
  - Defaults to a simple schema that models users and groups
  - Has read/write and read-only versions
- The *Custom DBMS Authenticator*:
  - Requires you to develop a custom plug-in class
  - May be applicable if the SQL Authenticator does not readily map to your user repository

Available Database Authenticators

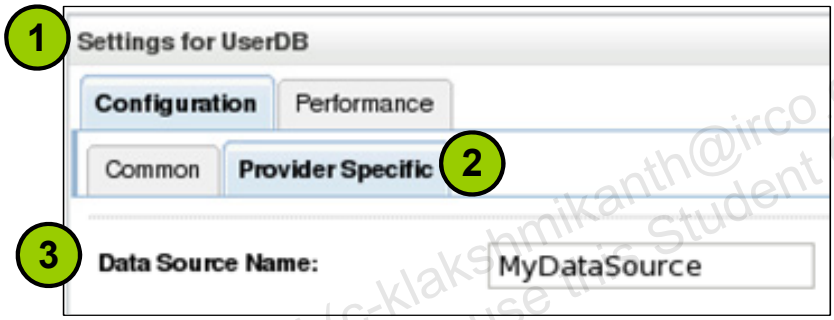
The SQL Authenticator uses a SQL database and allows both read and write access to the database. This authentication provider is configured by default with a typical SQL database schema, which you can configure to match your database's schema. The Read-Only SQL Authenticator includes only those attributes for retrieving users and groups. Therefore, this provider requires administrators to use a tool other than the administration console to create and update users and groups.

With the SQL Authenticator, specify the SQL statements used by the provider to access and edit the username, password, and group information in the database. The tables referenced by the SQL statements must exist in the database; the provider will not create them. You can modify these attributes as needed to match the schema of your database. However, if your database schema is radically different from this default schema, you may need to use a Custom DBMS authentication provider instead.

A Custom DBMS authentication provider requires that you write a plug-in class that implements the weblogic.security.providers.authentication.CustomDBMSAuthenticatorPlugin interface. The class must exist in the `CLASSPATH` and must be specified in the Plug-in Class Name attribute for the Custom DBMS authentication provider. Optionally, you can use the Plugin Properties attribute to specify values for properties defined by your plug-in class.

Authenticator Data Sources

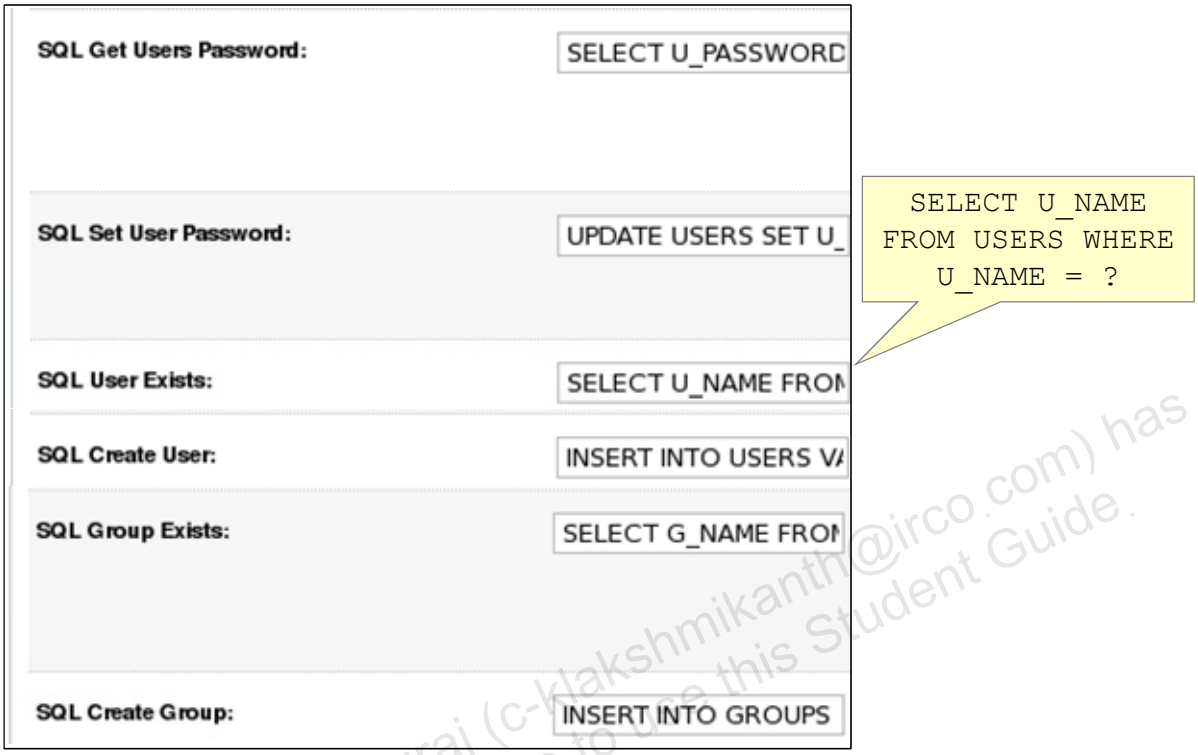
- Database providers require you to specify the name of an existing JDBC data source.
- The data source:
  - Provides the necessary connection parameters
  - Need not be explicitly targeted to servers



Authenticator Data Sources

1. After creating a new data source, locate and select your database authenticator in your security realm.
2. Click Configuration > Provider Specific.
3. Enter the name (not JNDI name) of your data source.
4. Edit additional provider attributes as necessary and click Save.

Configuring the SQL Authenticator

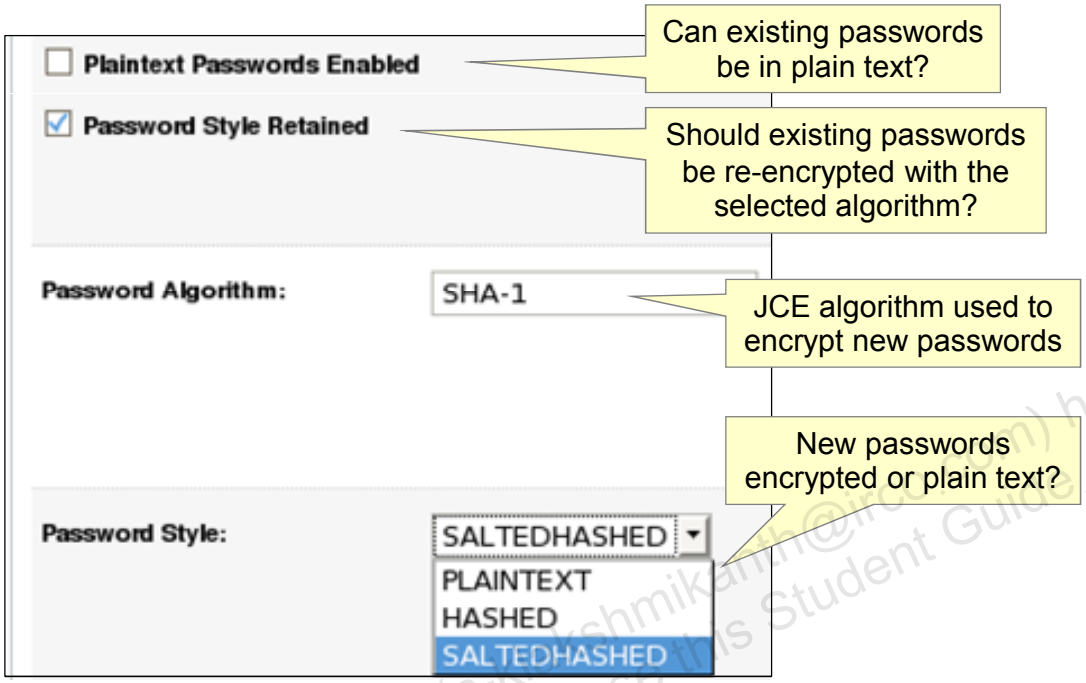


Configuring the SQL Authenticator

Enter the necessary SQL statements to create, read, update, and delete users and groups. Use "?" to indicate parameter values. The available attributes include:

- **SQL Get Users Password:** Look up a user's password given a username.
- **SQL Set User Password:** Set the password for a user. The statement requires two parameters: the password for the user and the username.
- **SQL User Exists:** Look up a user given a username.
- **SQL List Users:** Retrieve users that match a particular wildcard search. The statement requires a single parameter for the search expression.
- **SQL Create User:** Create a new user record. There are a minimum of two parameters: a username and its associated password. If Descriptions Supported is also true, the user's description is required as well.
- **SQL Remove User:** Delete a user given a username.
- **SQL Is Member:** Look up members of a group. The statement requires two parameters: a group name and a member name (user or group).
- **SQL Add Member To Group:** Add a specific member to a group. The statement requires two parameters: the group name and the group member being added.

Configuring the SQL Authenticator: Passwords



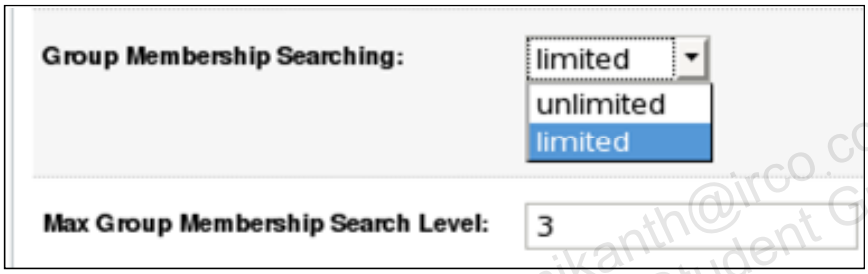
Configuring the SQL Authenticator: Passwords

Enter values for any of the following attributes that apply to password management:

- **Plaintext Passwords Enabled:** Specifies whether plaintext passwords are allowed to be used
- **Password Style Retained:** True indicates that the password style and algorithm that were in use for the original password in the database should be used for the new password. This setting is the default. False indicates the settings for Password Algorithm and Password Style will be used for the new password.
- **Password Algorithm:** The message digest algorithm used to hash passwords for storage. The name must be recognized by a Java Cryptography Extension (JCE) provider that is available at run time.
- **Password Style:** Indicates the password style that is used when storing passwords for users that are created and for changing the user's password if Password Style Retained is disabled. A hashed password has been encrypted using some function along with a known key. A salted password has been padded with randomly generated data prior to being hashed. Each generated salt value is typically stored along with the password for validation purposes.

Configuring the SQL Authenticator: Subgroups

- Group members may include other groups.
- To improve performance, you can limit the depth that the provider will search for subgroups.



Configuring the SQL Authenticator: Subgroups

The Group Membership Searching and Max Group Membership Search Level attributes specify whether recursive group membership searching is unlimited or limited, and if limited, how many levels of group membership can be searched. For example, if you specify that Group Membership Searching is LIMITED, and the Max Group Membership Search Level is 0, then the RDBMS authentication providers will find only groups that the user is a direct member of. Specifying a maximum group membership search level can greatly increase authentication performance in certain scenarios, because it may reduce the number of DBMS queries executed during authentication. However, you should only limit group membership search if you can be certain that the group memberships you require are within the search level limits you specify.

SQL Authenticator WLST Example

Add a SQL Authenticator to a realm:

```
edit()
startEdit()

realm = getMBean('/SecurityConfiguration/HRDomain/Realms/
   HRProdRealm')
provider = realm.createAuthenticationProvider('UserDB',
'weblogic.security.providers.authentication.SQLAuthenticator')

provider.setControlFlag('REQUIRED')
provider.setDataSourceName('MyDataSource')
provider.setGroupMembershipSearching('limited')
provider.setMaxGroupMembershipSearchLevel(3)
provider.setSQLUserExists(
   'SELECT U_NAME FROM USERS WHERE U_NAME = ?')
...
save()
activate(block='true')
```


SQL Authenticator WLST Example

Refer to the following MBeans in the documentation:

- `SecurityConfigurationMBean`
- `RealmMBean`
- `AuthenticationProviderMBean`
- `DBMSAuthenticatorMBean (a subclass of AuthenticationProviderMBean)`
- `SQLAuthenticatorMBean (a subclass of DBMSAuthenticatorMBean)`

Default Schema

The tables that correspond to the SQL statements must exist before using the authenticator.

Table Name	Column Names
USERS	U_NAME, U_PASSWORD, U_DESCRIPTION
GROUPS	G_NAME, G_DESCRIPTION
GROUPMEMBERS	G_NAME, G_MEMBER

Default Schema

The tables referenced by the SQL statements must exist in the database; the provider will not create them. You can modify these attributes as needed to match the schema of your database.

The user and group description columns are only required if the Descriptions Supported attribute is also true.

Section Summary

In this section, you should have learned how to:

- List the available database authenticators
- Configure SQL statements used to authenticate users and manage groups
- Initialize the default SQL Authenticator schema

Practice 16-2:
Authenticate Using a Database

This practice covers the following topics:

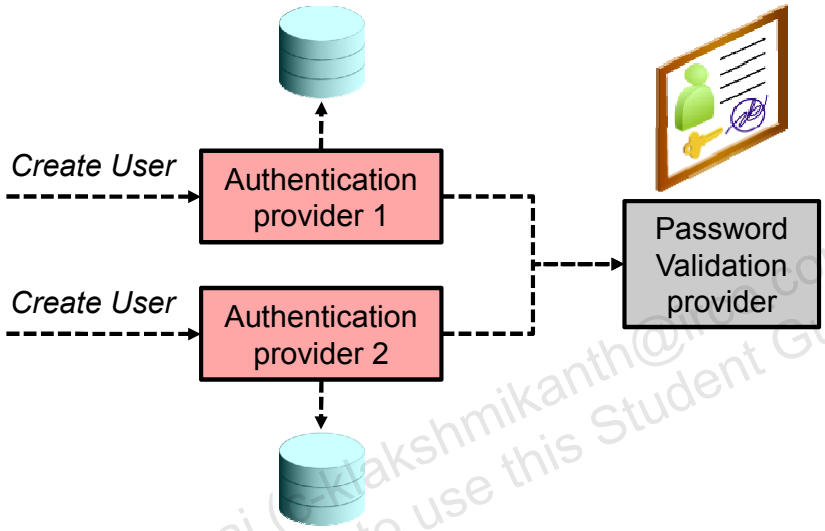
- Configuring a database to support the SQL Authenticator
- Adding a SQL Authenticator to a security realm
- Using the console to add users and groups to the database

Road Map

- WebLogic Authentication
- LDAP Providers
- Database Providers
- Password Validation
  - Validation Provider Architecture
  - Password Composition Rules
  - SystemPasswordValidator
- Security Migration

Password Validation Providers

When authentication providers are used to create or update users, they use the realm's password validation provider, if configured.



Password Validation Providers

WebLogic Server includes a Password Validation provider, which manages and enforces a set of configurable password composition rules. When configured in a security realm, the Password Validation provider is automatically invoked by a supported authentication provider whenever a password is created or updated for a user in that realm. The Password Validation provider then performs a check to determine whether the password meets the criteria established by the composition rules, and the password is accepted or rejected as appropriate.

Authentication Provider Support

- Authentication providers bundled with WebLogic Server use the password validator if they support user creation and/or password editing:
  - Default Authenticator
  - LDAP Authenticators (password editing only)
  - SQL Authenticator
- Custom providers may choose to use or not to use the password validator.
- Your user repository may also contain additional password validation logic.

Authentication Provider Support

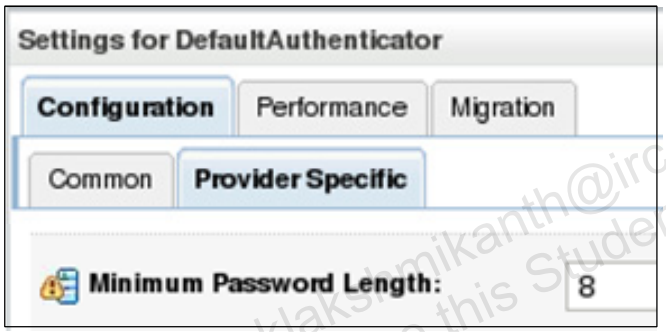
The following authentication providers can be used with the Password Validation provider:

- Default authentication provider
- SQL authenticator provider
- LDAP authentication provider
- Oracle Internet Directory Authentication Provider
- Oracle Virtual Directory Authentication Provider
- Active Directory authentication provider
- iPlanet authentication provider
- Novell authentication provider
- Open LDAP authentication provider

Default Authenticator and Password Validation

The Default Authenticator:

- Has an attribute to configure a password's minimum length
- Uses a password validation provider in addition to this rule, if one is available



Default Authenticator and Password Validation

By default, the Default authentication provider requires a minimum password length of eight characters. However, the minimum password length enforced by this provider can be customized. If the Default authentication provider and Password Validation provider are both configured in the security realm, and you attempt to create a password that does not meet the minimum length enforced by the Default authentication provider, an error is generated.

If the Default authentication provider rejects a password because it does not meet the minimum length requirement, the Password Validation provider is not called. To ensure that the Password Validator is always used with the Default authentication provider, make sure that the minimum password length is the same for both providers.

Password Composition Rules

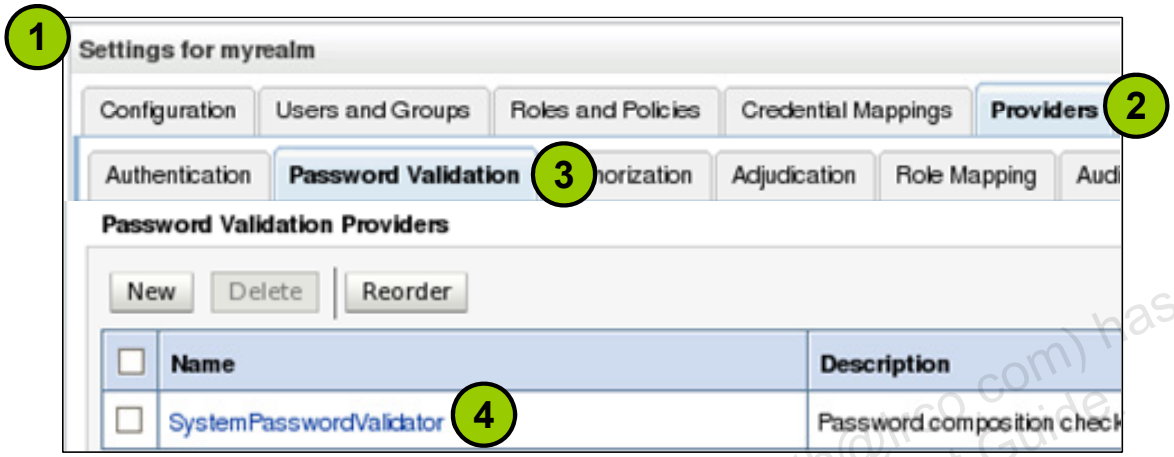
WebLogic Server includes a single provider implementation, which tests passwords against a set of composition rules, including:

- What is the length?
- How many alphabetic, numeric, and other characters are present?
- How many lowercase and uppercase characters are present?
- How many instances of the same character are present?
- Does the password contain the username?

Password Composition Rules

Setting password composition rules is only one component of hardening the WebLogic Server environment against brute-force password attacks. To protect user accounts, you should also configure user lockout. User lockout specifies the number of incorrect passwords that may be entered within a given interval of time before the user is locked out of his or her account.

Accessing the Password Validator



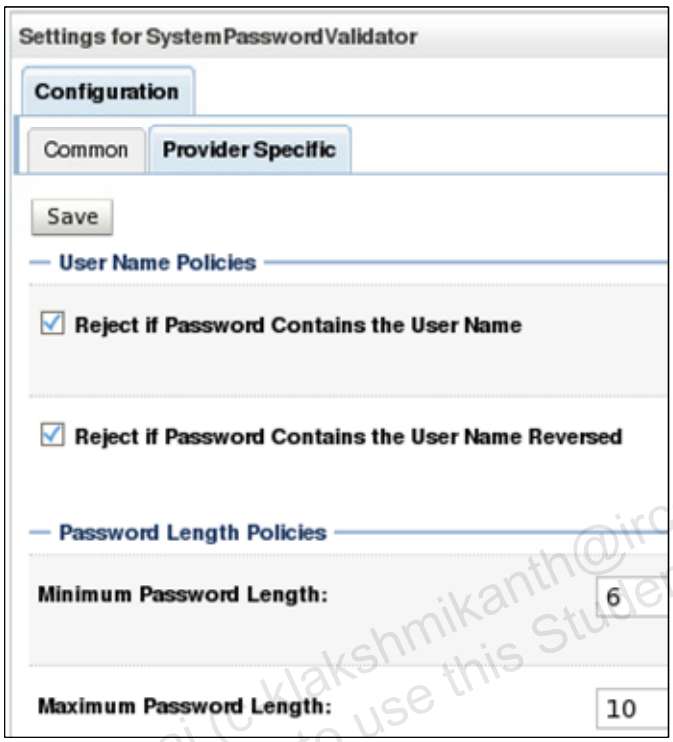
Accessing the Password Validator

Edit a security realm.

Click the Providers > Password Validation tab.

Select the default provider named SystemPasswordValidator. If it does not exist, click New.

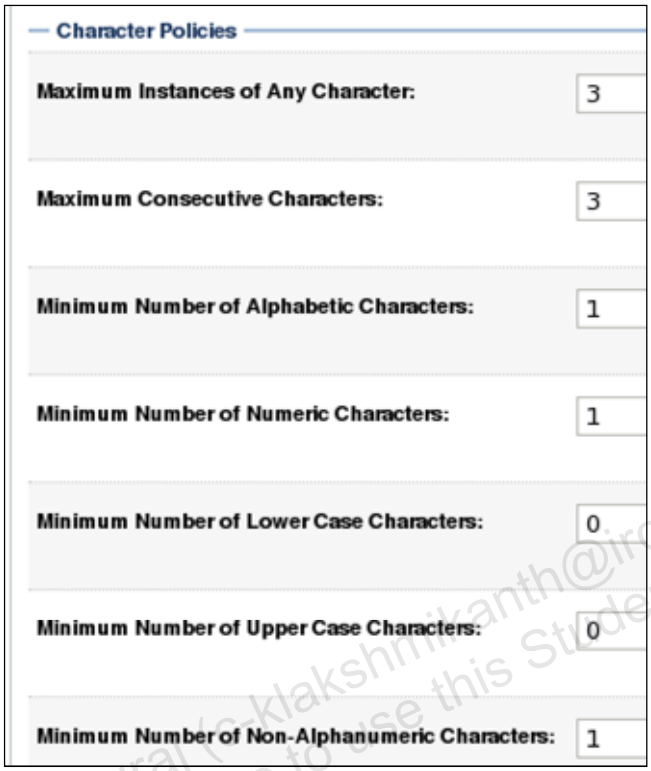
Configuring the Password Validator



Configuring the Password Validator

- **Reject if Password Contains the User Name:** Specifies whether the password can contain, or be set to, the username.
- **Reject if Password Contains the User Name Reversed:** Specifies whether the password can contain or be set to the reverse of the username.
- **Minimum Password Length:** Specifies the minimum number of characters that the password may contain. The recommended minimum value is 6.
- **Maximum Password Length:** Specifies the maximum number of characters that the password may contain. Specifying 0 results in no restriction on password length. The recommended value is 12.

Configuring the Password Validator



Configuring the Password Validator (continued)

- **Maximum Instances of Any Character:** Specifies the maximum number of times any one character may appear in the password. For example, if this value is set to 2, the password "alabaster" is rejected. Specifying 0 results in no restriction. The recommended value is 4.
- **Maximum Consecutive Characters:** Specifies the maximum number of times that a character may appear consecutively in the password. A value of 0 means no restriction. For example, if you specify a value of 2, a user cannot create a password such as "baaag.” The recommended value is 3.
- **Minimum Number of Alphabetic Characters:** Specifies the minimum number of alphabetic characters that a password must contain. The recommended value is 1.
- **Minimum Number of Lower Case Characters:** Specifies the minimum number of lowercase characters that a password must contain. The recommended value is 1.
- **Minimum Number of Non-Alphanumeric Characters:** Specifies the minimum number of nonalphanumeric characters (also known as special characters, such as %, *, or #) that must appear in the password. The recommended value is 1.
- **Minimum Number of Non-Alphabetic Characters:** Specifies the minimum number of numeric or special characters that a password must contain. The recommended value is 1.

Password Validator WLST Example

Updating the default password validator:

```
provider = getMBean('/SecurityConfiguration/HRDomain/Realms/
   HRProdRealm/PasswordValidators/SystemPasswordValidator')

provider.setRejectEqualOrContainUsername(true)
provider.setRejectEqualOrContainReverseUsername(true)
provider.setMinPasswordLength(5)
provider.setMaxPasswordLength(12)
provider.setMinAlphabeticCharacters(1)
provider.setMinNumericCharacters(1)
provider.setMinLowercaseCharacters(0)
provider.setMinUppercaseCharacters(0)
provider.setMinNonAlphanumericCharacters(1)
provider.setMaxConsecutiveCharacters(2)
provider.setMaxInstancesOfAnyCharacter(3)
```


Password Validator WLST Example

Refer to the following MBeans in the documentation:

- `SecurityConfigurationMBean`
- `RealmMBean`
- `PasswordValidatorMBean`
- `SystemPasswordValidatorMBean (a subclass of PasswordValidatorMBean)`

Section Summary

In this section, you should have learned how to:

- Describe the relationship between authentication and password validation security providers
- List several examples of password composition rules
- Configure the default password validation provider using the console or WLST

Practice 16-3:
Define Password Rules

This practice covers the following topics:

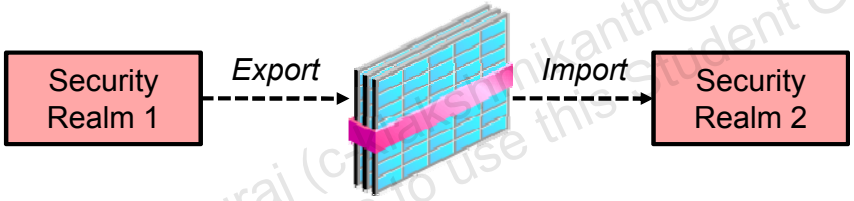
- Configuring password validation rules
- Creating users and verifying password rules

Road Map

- WebLogic Authentication
- LDAP Providers
- Database Providers
- Password Validation
- Security Migration
  - Migration Scenarios
  - Data Formats
  - Import/Export

Security Migration Scenarios

- Migration involves the copying of security artifacts (users, groups, roles, policies, and so on) from one realm to another.
- Migrate security data to:
  - Create snapshots for backup and recovery
  - Initialize a domain's security after creating it from a custom template
  - Transition from development/test environments to production
  - Synchronize development/test environments with the latest production security configuration



ORACLE

Security Migration Scenarios

WebLogic security realms persist different kinds of security data, including users and groups (for the Default authentication provider), security policies (for the XACML Authorization provider), security roles (for the XACML Role Mapping provider), and credential maps (for the WebLogic Credential Mapping provider). When you configure a new security realm or a new security provider, you may prefer to use the security data from your existing realm or provider, rather than re-create all the users, groups, policies, roles, and credential maps. Several WebLogic security providers support security data migration. This means that you can export security data from one security realm and import it into a new security realm.

While domain templates can include simple security role definitions within an XML file, any custom role or authorization policies in the source domain's security realm are not captured in a domain template. Instead, you must perform security migration as a separate task.

Migration may also be helpful to move data from one security realm to a new security realm in the same WebLogic Server domain, where one or more of the default WebLogic security providers will be replaced with new security providers.

Provider Migration Support

- Only those providers that use the embedded LDAP or database security store support migration using WebLogic Server.
- For providers that use some other external repository, use that software's migration capabilities.

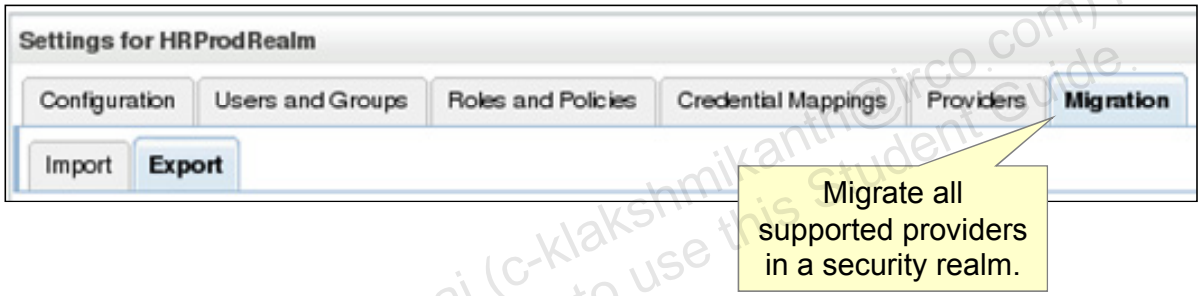
Provider Migration Support

The security providers that support migration via the console or WLST include:

- Default authentication provider
- Default Authorization provider
- XACML Authorization provider
- Default Role Mapping provider
- XACML Role Mapping provider
- Default Credential Mapping provider
- SAML Credential Mapping provider
- SAML identity assertion provider

Migration Approaches

- Export/import an entire security realm
- Export/import individual security providers
- Migrate the raw data from the realm persistent store by using a third-party tool:
  - LDAP Data Interchange Format (LDIF) files for the embedded LDAP
  - Database



Migration Approaches

You can migrate security data for each security provider individually or migrate security data for all the WebLogic security providers at once (that is, security data for an entire security realm). Note that you can only migrate security data from one provider to another if the providers use the same data format. You migrate security data through the WebLogic Administration Console or by using the WebLogic Scripting Tool (WLST).

Export files are the files to which security data is written (in the specified format) during the export portion of the migration process. Import files are files from which security data is read (also in the specified format) during the import portion of the migration process. Both export and import files are simply temporary storage locations for security data as it is migrated from one security provider's data store to another security provider's data store. The directory and file into which you export the security data should be carefully protected with operating system security as they contain secure information about your deployment.

Migration Formats

To migrate security data between two providers, they must both support the same migration format.

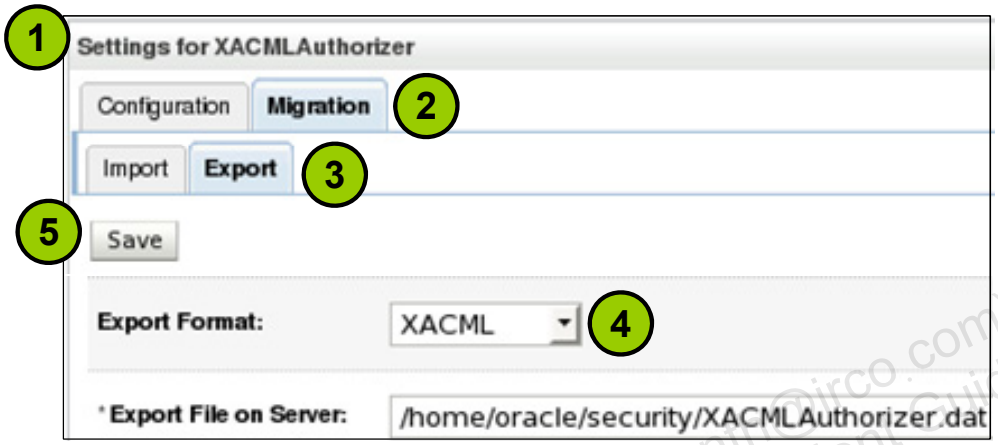
Provider	Supported Migration Formats
Default Authentication	WebLogic Authentication
Default Role Mapping	WebLogic Role
XACML Role Mapping	XACML 2.0 WebLogic Role
Default Authorization	WebLogic Authorization
XACML Authorization	XACML 2.0 WebLogic Authorization
Default Credential Mapping	WebLogic Credential
SAML Credential Mapping	WebLogic SAML + Java Key Store (JKS)

Migration Formats

In order for security data to be exported and imported between security providers, both security providers must process the same format. Some data formats used for the WebLogic Server security providers are unpublished; therefore, you cannot currently migrate security data from a WebLogic security provider to a custom security provider, or vice versa, using the unpublished formats.

Some providers also support various constraints. Constraints are key/value pairs that specify options to the export or import process. Use constraints to control which security data is exported to or imported from the security realm's store. For example, you may want to export only users and not groups from an authentication provider.

Migration Example: XACML Authorizer



Migration Example: XACML Authorizer

To export security data from a security provider to a file using the console:

1. In the left pane, select Security Realms and then select the name of the realm that you are configuring (for example, myrealm). Select the type of provider from which you want to export security data (for example, Authentication or Authorization). Then select the security provider from which you want to export security data.
2. Click the Migration tab.
3. Click the Export tab.
4. Indicate the Export Format. Then specify the directory and filename in which to export the security data in the Export File on Server field. The directory must exist. Optionally, define a specific set of security data to be exported in the Export Constraints box.
5. Click Save.

Security Migration WLST Example

- Export policies from an authorization provider:

```
domainRuntime()
provider = getMBean('DomainServices/DomainRuntimeService/
   DomainConfiguration/mydomain/SecurityConfiguration/
   mydomain/DefaultRealm/myrealm/Authorizers/XACMLAuthorizer')

provider.exportData('XACML',
   '/home/oracle/security/security.dat',None)
```

- Import policies into an authorization provider:

```
...
provider.importData('XACML',
   '/home/oracle/security/security.dat',None)
```

Security Migration WLST Example

Both `XACMLRoleMapperMBean` and `XACMLAuthorizerMBean` include the following operations:

- `exportData`: Exports provider-specific data in a specified format to the specified file and optionally using the given constraints (a Properties object)
- `exportResource`: Exports provider-specific policy data using an LDAP CN search filter to specify the resources for export
- `importData`: Imports provider-specific data in a specified format from the specified file and optionally using the given constraints (a Properties object)

Section Summary

In this section, you should have learned how to:

- Describe situations that require security migration
- Explain the WebLogic Server security migration process
- Export security assets from a realm or provider and import them into another

Quiz

Name four authentication providers supplied with WebLogic Server.

a. LDAP Authenticator
b. Principal Authenticator
c. OpenLDAP Authenticator
d. SQL Authenticator
e. Custom DBMS Authenticator
f. Schema Authenticator

Answer: a, c, d, e

Quiz

Which of the following is NOT an available authentication provider control flag?

a. SUFFICIENT
b. REQUISITE
c. OPTIONAL
d. ALWAYS
e. REQUIRED

Answer: d

Quiz

What two values can the Group Membership Searching authentication provider attribute be set to?

a. Unlimited
b. Restricted
c. Limited
d. Subtree

Answer: a, c

Lesson Summary

In this lesson, you should have learned how to:

- Explain WebLogic Server's security realm architecture
- Discuss basic LDAP concepts
- Configure and tune various authentication providers
- Use a password validation provider
- Migrate WebLogic users, groups, roles, or policies

17

Server Performance Essentials

Objectives

After completing this lesson, you should be able to:

- Describe a standard performance tuning methodology
- Run a load test by using The Grinder
- Tune some basic JVM settings
- Monitor JVM performance
- List some simple techniques that may improve server performance
- Create work managers and associate them with applications

Road Map

- Tuning Concepts
  - Terminology
  - Bottlenecks
  - Methodology
  - The Grinder Overview
- JVM Tuning
- WLS Tuning
- Work Managers

Performance Terminology

The performance of a software system is often characterized by its:

- **Response time:** A time metric (for example, the round-trip time it takes the server to deliver a Web page)
- **Throughput:** A rate metric (requests per unit of time) (for example, requests per second, bits per second)
- **Resource utilization:** A consumption metric (for example, the percent of CPU usage)
- **Scalability:** The ability of a system to achieve required performance as the load and system resources are both increased

Performance Terminology

The term “latency” often occurs in discussions about response time. In general, latency refers to the time between the issuing of a request and the time when the work actually begins on the request. For example, when a browser issues an HTTP request, it takes some amount of time for the request to travel over the network and arrive at the target server. That time spent in transit (while no work is being done on the request) is network latency.

Naturally, you will want to improve the response time, reduce the latency, increase the throughput, and optimize the resource utilization of the system. However, most often these factors are very interdependent, and it is rarely possible to tune one without affecting others.

The scalability of the system is related to the ability of the system to accommodate increasing load without having the system degrade unacceptably in performance.

Bottlenecks

- A CPU-bound system cannot process additional workload because the processor is too busy (at or near 100%).
- An I/O bound system's processor is not fully utilized (< 75%), even when the workload is increased; common culprits include:
  - Local or remote disks
  - Databases
  - Network bandwidth

Bottlenecks

Generally, your goal is to get to a point where the application server achieves your target CPU utilization. If you find that the application server CPU is under utilized, confirm whether the database is bottlenecked. If the database CPU is 100 percent utilized, then check your application SQL calls query plans. For example, are your SQL calls using indexes or doing linear searches? Also, confirm whether there are too many ORDER BY clauses used in your application that are affecting the database CPU. If you discover that the database disk is the bottleneck (for example, if the disk is 100 percent utilized), try moving to faster disks or to a RAID (redundant array of independent disks) configuration, assuming the application is not doing more writes then required.

Once you know the database server is not the bottleneck, determine whether the application server disk is the bottleneck. The disk I/O on an application server can be optimized using faster disks or RAID, disabling synchronous JMS writes, using JTA direct writes for transaction logs, or increasing the HTTP log buffer.

Check the amount of data transferred between the application and the application server, and between the application server and the database server. This amount should not exceed your network bandwidth; otherwise, your network becomes the bottleneck.

Benchmarking

- A performance benchmark:
  - Involves a realistic and repeatable test of your system
  - Establishes a set of baseline performance metrics to compare against
  - Helps set performance expectations
- Realistic tests should typically take into account:
  - Utilizing the same hardware, OS, and database configurations as the production system
  - Clients connecting over a WAN instead of a LAN
  - Avoiding “stubbed out” code

Benchmarking

To determine your performance objectives, you need to understand the application deployed and the environmental constraints placed on the system. Gather information about the levels of activity that components of the application are expected to meet, such as:

- The anticipated number of users
- The number and size of requests
- The amount of data and its consistency
- Your target CPU utilization

Performance objectives are limited by constraints, such as the configuration of hardware and software, including CPU type, disk size versus disk speed, and memory. There is no single formula for determining your hardware requirements. The process of determining what type of hardware and software configuration is required to meet application needs adequately is called capacity planning. Capacity planning requires assessment of your system performance goals and an understanding of your application. Capacity planning for server hardware should focus on maximum performance requirements. In addition, consider how much of your WebLogic Server machine’s processing power is consumed by processes unrelated to WebLogic Server. In the case in which WebLogic Server (or the machine on which it resides) is doing substantial additional work, you need to determine how much processing power will be drained by other processes.

Performance Testing Methodology

1. Define the expected workload.
2. Define performance objectives.
3. Select the subsystems to study.
4. Perform a test to create an initial benchmark.
5. Select one parameter to modify.
6. Perform another test.
7. Analyze the results.
8. Repeat steps 5–7.

Performance Testing Methodology

Performance tuning is not a silver bullet. Simply put, good system performance depends on good design, good implementation, defined performance objectives, and performance tuning. Similarly, performance tuning is an ongoing process. Implement mechanisms that provide performance metrics that you can compare against your performance objectives, allowing you to schedule a tuning phase before your system fails. The objective of this process is to meet your performance objectives, not to eliminate all bottlenecks. Resources within a system are finite. By definition, at least one resource (CPU, memory, or I/O) will be a bottleneck in the system. Tuning allows you to minimize the impact of bottlenecks on your performance objectives.

It is very important to make benchmarks realistic; otherwise, there will be unexpected results when the application goes into the production environment. If an application accesses a database, ensure that it accesses the database during the stress test. If you remove the database access, benchmark numbers will be meaningless.

In production, many clients might connect over a WAN. If so, ensure that you include testing client connections over a WAN during the stress test. Several products simulate WAN client connections. Finally, if real people test an application, the test results can be more realistic. That way you can see how the system will really be used when it goes into production.

Load Testing

- A **load test** is a benchmark taken while the system is handling a certain number of concurrent requests.
- A **stress test** is a type of load test designed to determine a system's limits.
- Load simulation tools generate a specific number of concurrent requests and provide resulting metrics.
- To maintain test integrity:
  - Always generate loads on separate machines from those hosting the system under test
  - Minimize the amount of data collection software running on the machines hosting the system
  - Allow the system to ramp up and self-tune before recording metrics

Load Testing

While functional testing verifies that an application demonstrates the correct behavior under certain inputs, load testing determines if an application can support a specified load (for example, 500 concurrent users) with specified response times. Load testing is used to create benchmarks.

Stress testing is load testing over an extended period of time. Stress testing determines whether an application can meet specified goals for stability and reliability, under a specified load, for a specified time period.

It is often possible to begin a stress-testing plan by taking the existing use cases for the application to be tested. Because use cases provide a description of how the application will typically be used when in production, they can often be translated directly into test scripts that can be run against the application.

The stress tests should include micro benchmarks, tests on small parts of the application. The parts of an application that are used the most should be tested on their own. That way, if you can tune those parts, your performance will increase quite a bit. Even though testing the most used parts of an application is important, it does not replace stress testing the whole application (including the database access). In the end, your entire application is going to be your best benchmark.

Load Testing Tools

Many different commercial and open-source load testing tools are available in the market, including:

- The Grinder
- JMeter
- OpenSTA
- HP LoadRunner
- RadView WebLOAD

Load Testing Tools

JMeter is an Apache Jakarta project that can be used as a load testing tool for analyzing and measuring the performance of a variety of services, with a focus on web applications. JMeter can be used as a unit test tool for JDBC database connections, FTP, LDAP, Web services, JMS, HTTP, and generic TCP connections. JMeter also supports assertions to ensure that the data received is correct, per thread cookies, configuration variables, and a variety of reports.

OpenSTA is a distributed software testing architecture designed around CORBA. The current toolset has the capability of performing scripted HTTP and HTTPS heavy load tests with performance measurements from Win32 platforms. Recordings are made in the tester's own browser, producing simple scripts that can be edited and controlled with a special high-level scripting language. These scripted sessions can then be played back to simulate many users by a high-performance load generation engine.

LoadRunner is a performance and load testing product by Hewlett-Packard (because it acquired Mercury Interactive in November 2006) for examining system behavior and performance, while generating actual load. Working in LoadRunner involves using three different tools which are part of LoadRunner. They are Virtual User Generator (VuGen), Controller, and Analysis.

The Grinder

- The Grinder:
  - Is a load testing tool
  - Is an open source implementation based on Java and Python
  - Supports a distributed, agent-based load simulation model
  - Provides a graphical console to manage agents and view aggregated results
  - Supports HTTP/S (links, forms, cookies, and so on) but can be extended to support additional protocols
- A Grinder agent:
  - Utilizes a specific number of processes and threads
  - Executes a supplied test script with each thread

The Grinder

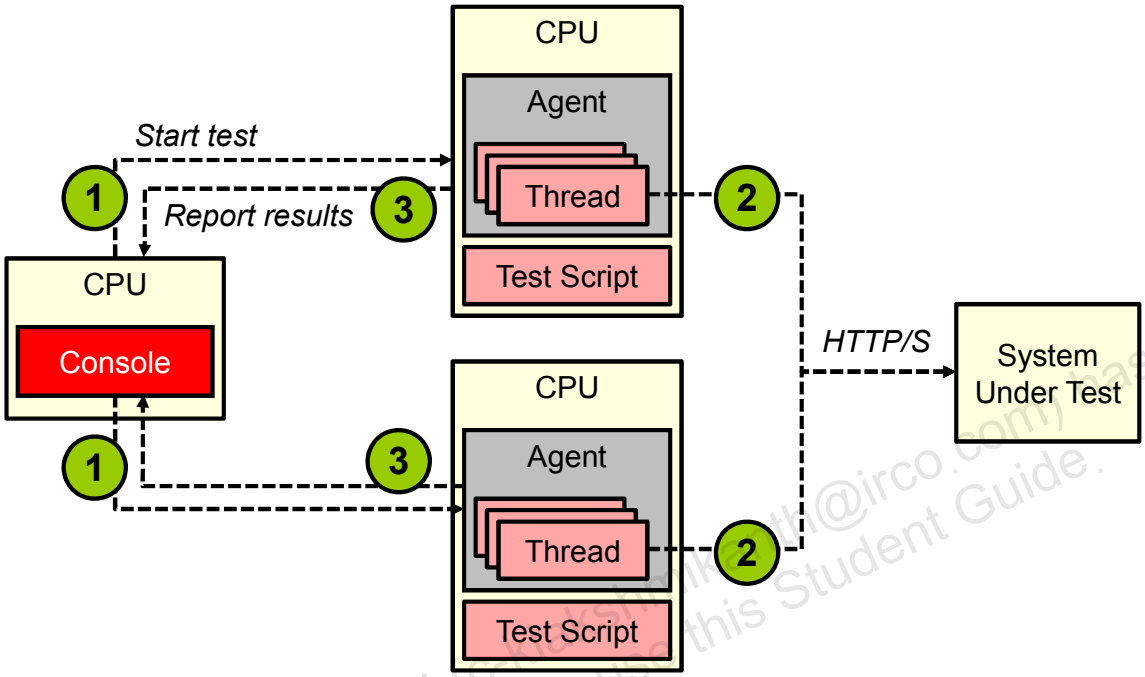
The Grinder is a Java load testing framework that makes it easy to run a distributed test using many load injector machines. It is freely available under a BSD-style open-source license, and is based on other open source technologies such as Jython, HTTPClient, XMLBeans, and PicoContainer. The Grinder was originally developed for the book *Professional Java 2 Enterprise Edition with BEA WebLogic Server*.

Each test context runs in its own thread. The threads can be split over many processes depending on the requirements of the test and the capabilities of the load injection machine. The Grinder makes it easy to coordinate and monitor the activity of processes across a network of many load injection machines from a central console.

Scripts can be created by recording actions of a real user by using The TCPProxy. The script can then be customized by hand. Input data (for example, URL parameters or form fields) can be dynamically generated. The source of the data can be anything including flat files, random generation, a database, or previously captured output.

The Grinder has special support for HTTP that automatically handles cookie and connection management for test contexts. Users can write their own plug-ins to a documented interface, although this is rarely necessary due to the powerful scripting facilities.

The Grinder Architecture



The Grinder Architecture

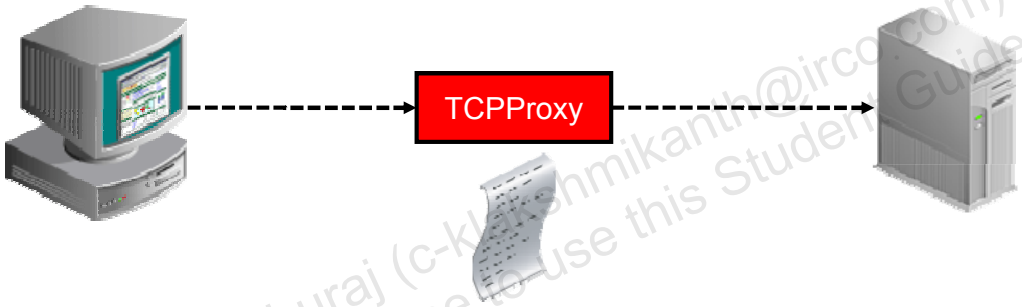
The Grinder is a framework for running test scripts across a number of machines. The framework is comprised of three types of processes (or programs): worker processes, agent processes, and the console. Worker processes interpret Jython test scripts and perform tests using a number of worker threads. Agent processes manage worker processes. The console coordinates the other processes, and collates and displays resulting statistics. Because The Grinder is written in Java, each of these processes is a Java Virtual Machine (JVM).

For heavy duty testing, you start an agent process on each of several load injector machines. The worker processes that they launch can be controlled and monitored using the console. There is little reason to run more than one agent on each load injector, but you can if you wish. Each worker process sets up a network connection to the console to report statistics. Each agent process sets up a connection to the console to receive commands, which it passes on to its worker processes. The console listens for both types of connection on a particular address and port.

A test is a unit of work against which statistics are recorded. Tests are uniquely defined by a test number and also have a description. Users specify which tests to run by using a Jython test script. The script is executed many times in a typical testing scenario. Each worker process has a number of worker threads, and each worker thread calls the script a number of times. A single execution of a test script is called a “run.”

The Grinder Proxy

- The Grinder tests are Python scripts, which can be:
  - Coded manually
  - Recorded using the TCPProxy
- Configure your Web browser to proxy requests through the TCPProxy.
- The resulting script will include all GET and POST requests, cookies, and user “think times.”



The Grinder Proxy

The TCPProxy is a proxy process that you can place in a TCP stream, such as the HTTP connection between your browser and a server. It filters the request and response streams, sending the results to the terminal window (`stdout`). You can control its behavior by specifying different filters. The TCPProxy’s main purpose is to automatically generate HTTP test scripts that can be replayed with The Grinder’s HTTP plug-in. Because the TCPProxy lets you see what is going on at a network level, it is also very useful as a debugging tool in its own right.

The TCPProxy appears to your browser just like any other HTTP proxy server, and you can use your browser as normal. If you open `http://grinder.sourceforge.net` with your browser, it displays the The Grinder home page, and the TCPProxy outputs all the HTTP interactions between the browser and the SourceForge site. It is important to remember to remove any “bypass proxy server” or “No proxy for” settings that you might have, so that all the traffic flows through the TCPProxy and can be captured.

Having finished your run-through, click Stop on the TCPProxy console and the generated script will be written to `grinder.py`. The `grinder.py` file contains headers, requests, and a logical grouping of requests into pages, for the recorded tests. The recorded script can also be edited manually to suit your needs.

Agent Properties

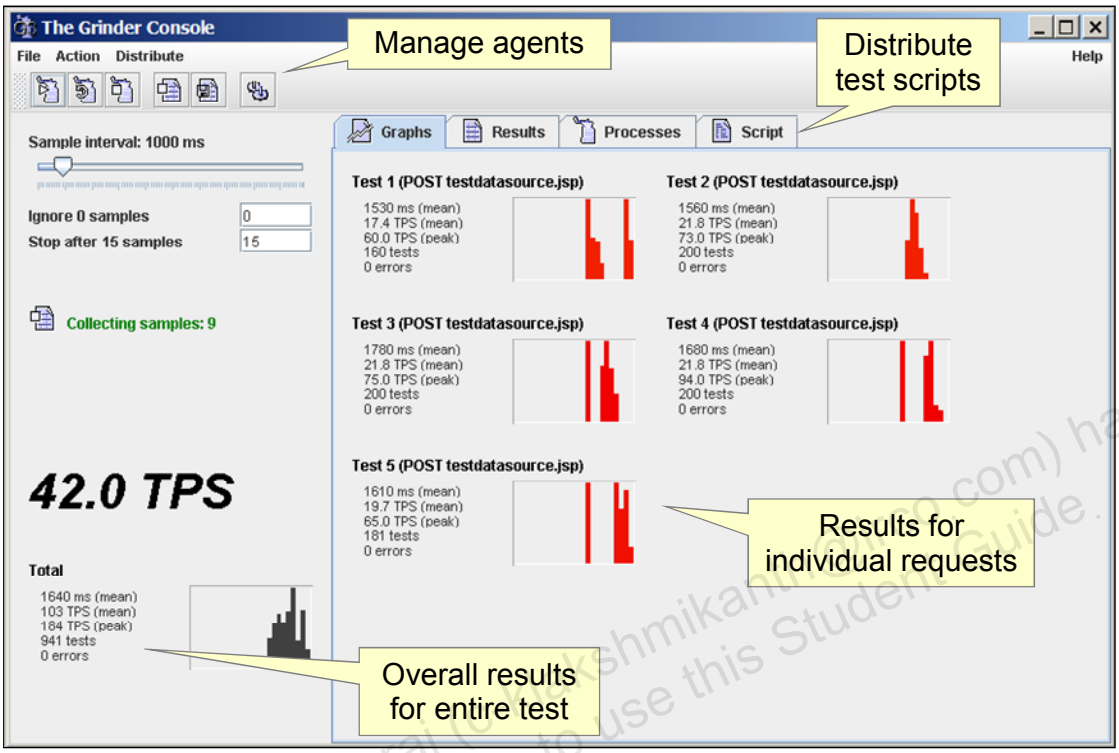
- Agents are configured using the `grinder.properties` file, which has settings for:
  - The location of the console
  - The test script to run
  - The number of processes to start
  - The number of threads to start in each process
  - The number of times that each thread should run the test
  - Think time adjustments (speed up or slow down)
  - Output and logging levels
- If the console is not available, agents run immediately.
- Test script files can also be distributed to agents by using the console.

Agent Properties

The Grinder worker and agent processes are controlled by setting properties in the `grinder.properties` file. All properties have default values. If you start a Grinder agent process without a `grinder.properties` file, the agent communicates with the console by using default addresses and use one worker process, one thread, and make one run through the test script found in the grinder.py file. The available properties include:

- `grinder.processes`: The number of worker processes that the agent should start
- `grinder.threads`: The number of worker threads that each worker process spawns
- `grinder.runs`: The number of runs of the test script that each thread performs. A value of 0 means “run forever,” and should be used when you are using the console to control your test runs.
- `grinder.processIncrement`: If set, the agent will ramp up the number of worker processes, starting the number specified every grinder.processesIncrementInterval milliseconds. The upper limit is set by `grinder.processes`.
- `grinder.duration`: The maximum length of time in milliseconds that each worker process should run. grinder.duration can be specified in conjunction with `grinder.runs`, in which case the worker processes will terminate if either the duration time or the number of runs is exceeded.

The Grinder Console



The Grinder Console

The Start processes, Reset processes, and Stop processes menu items send signals to The Grinder processes that are listening. These controls are disabled if no agents are connected to the console. On the Processes tab, you can check whether any agents are connected.

The Start processes control signals to worker processes that they should move into the running state. Processes that are already running ignore this signal. Processes that are in the finished state exit. The agent process then rereads the properties file, and launches new worker processes in the running state. The Reset processes control signals all the worker processes to exit. The agent process then rereads the properties file and launches new worker processes.

The sample controls determine how the console captures reports from the worker processes. It is important to understand that these control only the console behavior. They do not adjust the frequency at which the worker processes send reports. The slider controls the period at which the console takes a sample. This involves adding up all the reports received over that sample interval and calculating the TPS as (number of tests that occurred)/(interval length). It is also the period at which the console graphs and statistics are updated.

Each time the worker processes run, they generate a new set of logs. Logs from previous runs are “archived” by renaming them. The number of logs that are kept from previous runs can be controlled with `grinder.numberOfOldLogs`.

Section Summary

In this section, you should have learned how to:

- Define basic terms such as throughput, scalability, and CPU bound
- Describe a general methodology for performance testing
- List the capabilities of The Grinder tool
- Explain The Grinder architecture

Road Map

- Tuning Concepts
- JVM Tuning
  - JRockit Heap Settings
  - Sun JVM Heap Settings
  - Low Memory Detection
  - Heap Monitoring
  - JRockit Mission Control Overview
- WLS Tuning
- Work Managers

Java Virtual Machine (JVM) Selection

- WebLogic Server (WLS) relies on its host JVM to manage memory and I/O.
- The JVM and its configuration settings can significantly affect performance.
  - Oracle JRockit JVM
  - Sun Hotspot JVM
  - IBM JVM
  - HP JVM
- Oracle recommends that you use:
  - Oracle JRockit JVM for production servers
  - Sun Hotspot JVM for development servers and for running other WLS utilities

Java Virtual Machine (JVM) Selection

The Java Virtual Machine (JVM) is a virtual “execution engine” instance that executes the bytecodes in compiled Java class files on a microprocessor. How you tune your JVM affects the performance of WebLogic Server and your applications. Use only production JVMs on which WebLogic Server has been certified. Check the documentation for the latest list of JVMs and operating systems. The current release of WebLogic Server supports only those JVMs that are J2SE 5.0-compliant.

To the JVM, a stream of bytecodes is a sequence of instructions. Each instruction consists of a one-byte opcode and zero or more operands. The opcode tells the JVM what action to take. If the JVM requires more information to perform the action than just the opcode, the required information immediately follows the opcode as operands.

Tuning the JVM to achieve optimal application performance is one of the most critical aspects of WebLogic Server performance. A poorly tuned JVM can result in slow transactions, long latencies, system freezes, and even system crashes. Ideally, tuning should occur as part of the system startup, by employing various combinations of the startup options.

Setting WLS JVM Arguments

- Use the JAVA_VENDOR environment variable to select the Sun or Oracle JVM used by a specific server.
- Use the USER_MEM_ARGS variable to supply JVM arguments for a specific server.

Start a server with custom JVM settings:

```
export JAVA_VENDOR="Oracle"
export USER_MEM_ARGS="-Xms512m -Xmx1g"
./startWebLogic.sh
```


Setting WLS JVM Arguments

When you create a domain, if you choose to customize the configuration, the Configuration Wizard presents a list of SDKs that WebLogic Server installed. From this list, you choose the JVM that you want to run your domain, and the wizard configures the Oracle start scripts based on your choice.

After you create a domain, if you want to use a different JVM, you can modify the scripts and change either the JAVA_HOME or JAVA_VENDOR environment variables. For JAVA_HOME, specify an absolute pathname to the top directory of the SDK that you want to use. For JAVA_VENDOR, specify the vendor of the SDK. Valid values depend on the platform on which you are running:

- “Oracle” indicates that you are using the JRockit SDK. It is valid only on platforms that support JRockit.
- “Sun” indicates that you are using the Sun SDK.
- “HP” and “IBM” indicate that you are using SDKs that Hewlett Packard or IBM have provided. These values are valid only on platforms that support HP or IBM SDKs.

JRockit Review

JRockit:

- Is highly optimized for running server-side, high-throughput applications
- Is tuned for environments in which the JVM is restarted infrequently
- Supports Intel architectures on Windows and Linux
- Supports Solaris/SPARC
- Supports 32- and 64-bit architectures
- Employs adaptive optimizations at run time, such as compiling commonly used Java code into native code
- Provides a suite of management, tuning, and troubleshooting tools

JRockit Review

JRockit JVM is a high-performance JVM developed to ensure reliability, scalability, manageability, and flexibility for Java applications. JRockit JVM delivers better performance for Java applications deployed on Intel 32-bit (Xeon) and 64-bit (Xeon, Itanium, and SPARC) architectures at significantly lower costs to the enterprise. Furthermore, it is the only enterprise-class JVM optimized for Intel architectures, providing interoperability among multiple hardware and operating configurations. JRockit JVM makes it possible to gain optimal performance for your Java applications when running it on either the Windows or Linux operating platforms on either 32-bit or 64-bit architectures. JRockit JVM is especially well suited for running Oracle WebLogic Server.

The JRockit JDK is very similar, in the file layout, to the Sun JDK, except that it includes a new JRE with JRockit JVM and some changes to the Java class libraries (however, all of the class libraries have the same behavior in the JRockit JDK as in the Sun JDK).

JVM Tuning Parameters

Commonly tuned JVM settings include:

- Memory management (heap)
- Garbage collection
- Class loading
- Compilation/interpretation

JVM Tuning Parameters

The goal of tuning your heap size is to minimize the time that your JVM spends doing garbage collection while maximizing the number of clients that WebLogic Server can handle at a given time. To ensure maximum performance during benchmarking, you might set high heap size values to ensure that garbage collection does not occur during the entire run of the benchmark.

Depending on which JVM you are using, you can choose from several garbage collection schemes to manage your system memory. For example, some garbage collection schemes are more appropriate for a given type of application. Once you have an understanding of the workload of the application and the different garbage collection algorithms utilized by the JVM, you can optimize the configuration of the garbage collection.

Java Heap

- A JVM's heap is the area of memory where server and application objects live.
- Java programs are not responsible for directly allocating and freeing memory in the heap.
- Garbage collection (GC):
  - Is the process of freeing up memory used by objects that are no longer referenced by the program
  - Is automatically performed by the JVM
  - Typically defragments or "compacts" free memory as well
  - Helps to avoid memory leaks caused by programs directly managing memory
  - Requires CPU cycles and can therefore significantly impact program performance, if not properly tuned

Java Heap

The Java heap is where the objects of a Java program live. It is a repository for live objects, dead objects, and free memory. When an object can no longer be reached from any pointer in the running program, it is considered "garbage" and ready for collection. A best practice is to tune the time spent doing garbage collection to within 5% of execution time.

The JVM heap size determines how often and how long the VM spends collecting garbage. An acceptable rate for garbage collection is application-specific and should be adjusted after analyzing the actual time and frequency of garbage collections. If you set a large heap size, full garbage collection is slower, but it occurs less frequently. If you set your heap size in accordance with your memory needs, full garbage collection is faster, but occurs more frequently.

In addition to freeing unreferenced objects, a garbage collector may also combat heap fragmentation. Heap fragmentation occurs through the course of normal program execution. A second advantage of garbage collection is that it helps ensure program integrity. Garbage collection is an important part of Java's security strategy. Java programmers are unable to accidentally (or purposely) crash the JVM by incorrectly freeing memory.

WLS Production Recommendations for Any JVM

- Set the initial and maximum heap sizes to the same value.
- Increase the heap size until the performance test no longer produces Out-of-Memory exceptions.
- Do not set the combined heap sizes of all servers larger than 75% of available RAM, to avoid paging and long GC pauses.
- Avoid scenarios in which other systems will contend with WLS for the same RAM.
- If not CPU bound and more RAM is available, add another server to the cluster.

WLS Production Recommendations for Any JVM

In production environments, set the minimum heap size and the maximum heap size to the same value to prevent wasting VM resources used to constantly grow and shrink the heap. This also applies to the New generation size settings on the Sun JVM or the Nursery size settings on JRockit JVM. Although JRockit provides automatic heap resizing heuristics, they are not optimal for all applications.

The heap sizes should be set to values such that the maximum amount of memory used by the VM does not exceed the amount of available physical RAM. If this value is exceeded, the OS starts paging and performance degrades significantly. The VM always uses more memory than the heap size. The memory required for internal VM functionality, native libraries outside of the VM, and permanent generation memory (for the Sun VM only: memory required to store classes and methods) is allocated in addition to the heap size settings.

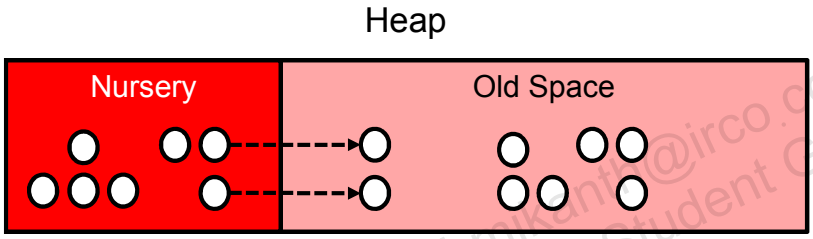
You might see the following Java error at run time if you are running out of heap space:

```
java.lang.OutOfMemoryError <<no stack trace available>>
```

If you find that you have a large amount of available free RAM remaining, run more instances of WebLogic Server on your machine.

Generational Garbage Collection

- JRockit supports standard and generational heaps.
- Heap memory is divided into areas called “generations,” which contains objects of different ages.
- Most generational garbage collection algorithms are based on the premise that most objects do not live long.



Generational Garbage Collection

The heap is sometimes divided into two areas (or generations) called the nursery (or young space) and the old space. The nursery is a part of the heap reserved for allocation of new objects. When the nursery becomes full, garbage is collected by running a special young collection, where all objects that have lived long enough in the nursery are promoted (moved) to the old space, thus freeing up the nursery for more object allocation. When the old space becomes full, garbage is collected there, a process called an old collection.

The reasoning behind a nursery is that most objects are temporary and short lived. A young collection is designed to be swift at finding newly allocated objects that are still alive and moving them away from the nursery. Typically, a young collection frees a given amount of memory much faster than an old collection or a garbage collection of a single-generational heap (a heap without a nursery).

In R27.2.0 and later releases, a part of the nursery is reserved as a keep area. The keep area contains the most recently allocated objects in the nursery and is not garbage-collected until the next young collection. This prevents objects from being promoted just because they were allocated right before a young collection started.

Generational Garbage Collection (continued)

During object allocation, JRockit JVM distinguishes between small and large objects. The limit for when an object is considered large depends on the JVM version, the heap size, the garbage collection strategy and the platform used, but is usually somewhere between 2 and 128 KB. See the documentation for -XXtlaSize and -XXlargeObjectLimit for more information.

Small objects are allocated in thread local areas (TLAs). The thread local areas are free chunks reserved from the heap and given to a Java thread for exclusive use. The thread can then allocate objects in its TLA without synchronizing with other threads. When the TLA becomes full, the thread simply requests a new TLA. The TLAs are reserved from the nursery if such exists, otherwise they are reserved anywhere in the heap.

Large objects that do not fit inside a TLA are allocated directly on the heap. When a nursery is used, the large objects are allocated directly in old space. Allocation of large objects requires more synchronization between the Java threads, although JRockit JVM uses a system of caches of free chunks of different sizes to reduce the need for synchronization and improve the allocation speed.

By default, JRockit JVM uses a dynamic garbage collection mode that automatically selects a garbage collection strategy to use, aiming at optimizing the application throughput. You can also choose between two other dynamic garbage collection modes or select the garbage collection strategy statically.

JRockit JVM uses two compaction methods called external compaction and internal compaction. External compaction moves (evacuates) the objects within the compaction area to free positions outside the compaction area and as far down in the heap as possible. Internal compaction moves the objects within the compaction area as far down in the compaction area as possible, thus moving them closer together. The JVM selects a compaction method depending on the current garbage collection mode and the position of the compaction area. External compaction is typically used near the top of the heap, while internal compaction is used near the bottom where the density of objects is higher.

Note that the JVM uses more memory than just the heap. For example, Java methods, thread stacks, and native handles are allocated in memory separate from the heap, as well as JVM internal data structures.

Basic JRockit JVM Arguments

Argument	Description
-Xms	The initial amount of heap allocated to the JVM ('m'=megabytes, 'g'=gigabytes)
-Xmx	The maximum amount of heap that this JVM can allocate
-Xns	Size of the nursery generation in the heap
-XgcPrio	A priority level that helps to determine which GC algorithms the JVM will use at run time: • throughput: Maximize application throughput • pausetime: Minimize how long GC runs • deterministic: Consistent response times
-XXcompact Ratio	The percentage of the heap that should be compacted during each GC cycle

Basic JRockit JVM Arguments

The `-Xms` option sets the initial and minimum Java heap size. Combine `-Xms` with a memory value and add a unit. If you do not add a unit, you will get the exact value you state (for example, 64 will be interpreted as 64 bytes, not 64 megabytes or 64 kilobytes).

The `-Xmx` option sets the maximum Java heap size. Depending upon the kind of operating system that you are running, the maximum value that you can set for the Java heap can vary. Note, however, that this option does not limit the total amount of memory that the JVM can use.

The `-Xns` option sets the nursery size. JRockit JVM uses a nursery when the generational garbage collection model is used. You can also set a static nursery size when running a dynamic garbage collector (`-XgcPrio`).

The `-XgcPrio` option sets a dynamic garbage collection mode. This garbage collector combines all types of garbage collection heuristics and optimizes the performance accordingly. When running this garbage collector, you only need to determine whether your application responds best to optimal memory throughput during collection or minimized pause times. The dynamic garbage collector will then adapt its choice of collector type, in run time, to what best suits your application.

Basic JRockit JVM Arguments (continued)

The `-XXcompactRatio` option sets the compaction ratio. Compaction is the garbage collector's main weapon against fragmentation. The idea is to look at a part of the heap and move all the live objects in that part together to create larger consecutive areas of free space. While the JVM is compacting the heap, all threads that want to access objects have to stand still, because the JVM is moving the objects around. Consequently, only a part of the heap is compacted to reduce pause time. In some cases, when the garbage collection time is too long, it might be beneficial to reduce the compaction area to reduce the pause times. In some other cases, especially when you are allocating very large arrays, it may be necessary to increase the compaction area to reduce the fragmentation on the heap and thus make allocation faster.

Nonstandard, or `-X`, command-line options are options that are exclusive to Oracle JRockit JVM that change the behavior of JRockit JVM to better suit the needs of different Java applications. These options are all preceded by `-X` and will not work on other JVMs (conversely, the nonstandard options used by other JVMs will not work with JRockit JVM).

Advanced `-XX` options are subject to change without notice and are recommended only if you have a thorough understanding of the JVM. If used improperly, these options can have negative effect on the stability or performance of your system.

JRockit Recommendations for WLS

- Allow JRockit to perform optimizations under load before collecting performance metrics.
- Use the default GC priority level (throughput)
- Set the nursery to approximately 25–40% of the heap size.
- Some application frameworks produce a lot of temporary objects; in this case, increase the nursery size.

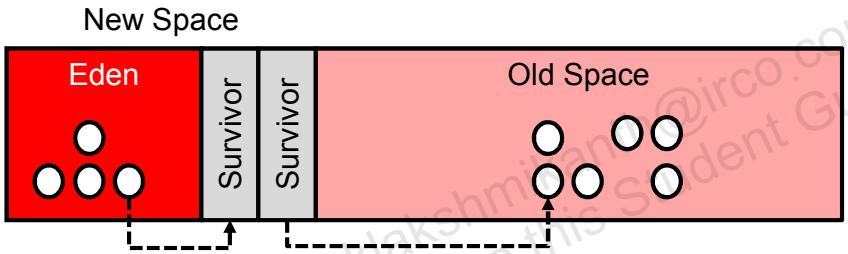
JRockit Recommendations for WLS

Oracle recommends setting the minimum heap size (`-Xms`) equal to the maximum heap size (`-Xmx`) to minimize garbage collections. Setting a low maximum heap value compared to the amount of live data decreases performance by forcing frequent garbage collections.

Optimally, you should try to make the nursery as large as possible while still keeping the garbage collection pause times acceptably low. This is particularly important if your application is creating a lot of temporary objects. The maximum size of a nursery cannot exceed 95% of the maximum heap size.

Sun Hotspot JVM Generational GC

- The heap is divided into these generations:
  - New Space
  - Old Space
  - Permanent
- New Space is further divided into an Eden area and two Survivor areas.



Sun Hotspot JVM Generational GC

Memory in the Java HotSpot virtual machine is organized into three generations: a young generation, an old generation, and a permanent generation. Most objects are initially allocated in the young generation. The old generation contains objects that have survived some number of young generation collections, as well as some large objects that may be allocated directly in the old generation. The permanent generation holds objects that the JVM finds convenient to have the garbage collector manage, such as objects describing classes and methods, as well as the classes and methods themselves.

The young generation consists of an area called Eden, plus two smaller Survivor spaces. Most objects are initially allocated in Eden. (As mentioned, a few large objects may be allocated directly in the old generation.) The Survivor spaces hold objects that have survived at least one young generation collection and have thus been given additional chances to die before being considered “old enough” to be promoted to the old generation. At any given time, one of the Survivor spaces holds such objects, while the other is empty and remains unused until the next collection.

When the young generation fills up, a young generation collection (sometimes referred to as a minor collection) of just that generation is performed. When the old or permanent generation fills up, what is known as a full collection (sometimes referred to as a major collection) is typically done.

Basic Sun JVM Arguments

Argument	Description
-Xms	The initial amount of heap
-Xmx	The maximum amount of heap
-XX:NewSize	The initial size of the New generation in the heap
-XX:MaxNewSize	The maximum size of the New generation
-XX:Survivor Ratio	The ratio of the Eden area to one Survivor area. For example, 8 means that the Survivor areas use 20% of the New generation space.

Basic Sun JVM Arguments

- `-XX:NewSize` (default 2 MB): Default size of new generation (in bytes)
- `-XX:MaxNewSize`: Maximum size of new generation (in bytes). Since 1.4, MaxNewSize is computed as a function of NewRatio.
- `-XX:NewRatio` (default=2): Ratio of new to old generation sizes
- `-XX:SurvivorRatio` (default=8): Ratio of Eden size to one Survivor space size.
- `-XX:TargetSurvivorRatio` (default=50%): Desired percentage of Survivor space used after scavenge
- `-XX:MaxPermSize`: Size of the permanent generation

Sun JVM Recommendations for WLS

- Set the initial and maximum New Space sizes to the same value.
- Set the New Space to approximately 25% of the heap size.
- Increase the New Space as the number of available CPUs increases.
- Use a Survivor ratio of 8.

Sun JVM Recommendations for WLS

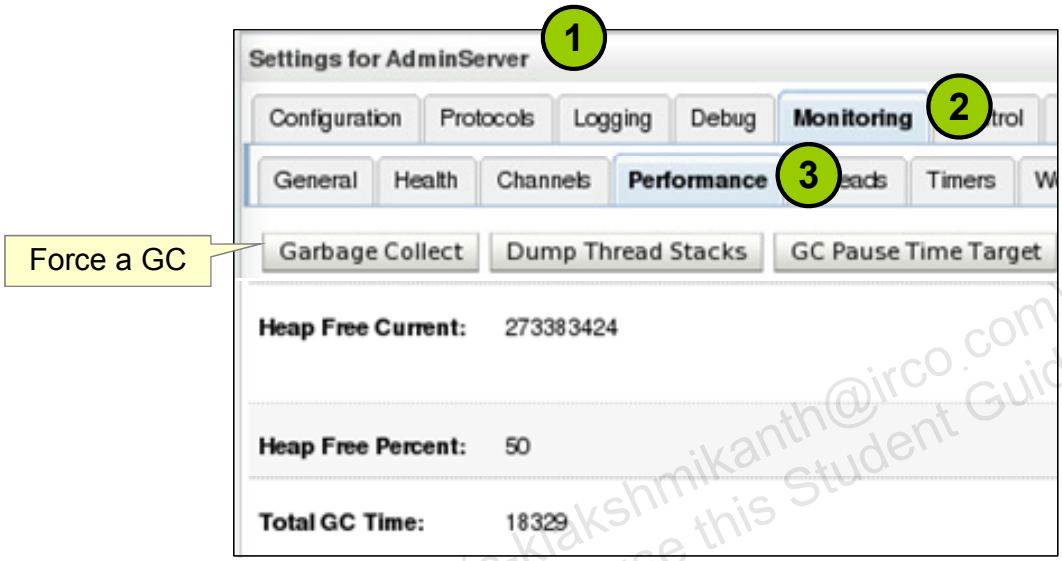
Test both the client and server HotSpot JVMs to see which options perform better for your particular application. The JVM documentation provides information on the command-line options and environment variables that can affect the performance characteristics of the HotSpot JVM.

As a general rule, set `-XX:NewSize` to be one-fourth the size of the heap size. Increase the value of this option for larger numbers of short-lived objects. Be sure to increase the New generation as you increase the number of processors. Memory allocation can be parallel, but garbage collection is not parallel.

Configure the ratio of the Eden/Survivor space size. Try setting this value to 8, and then monitor your garbage collection.

Monitoring a WLS JVM

The WLS console lets you monitor basic JVM metrics.



Monitoring a WLS JVM

1. Select a server in your domain.
2. Click the Monitoring tab.
3. Click the Performance tab to view various settings and statistics for the underlying server JVM. You can also use this page to force garbage collection or a thread dump.

Available statistics include Heap Free Current, Heap Size Current, Heap Free Percent, Total Physical Memory, Free Physical Memory, GC Algorithm, Total GC Count, and Last GC Start.

The GC Pause Time Target button sets a pause time target value for pause time or deterministic dynamic garbage collection. The target value is used as a pause time goal. This helps to more precisely configure the dynamic garbage collector to keep pauses near the target value. Using this option you can specify pause target values between 1 ms and 5 seconds. Also see the `-XpauseTarget` command-line argument.

WLS Low Memory Detection

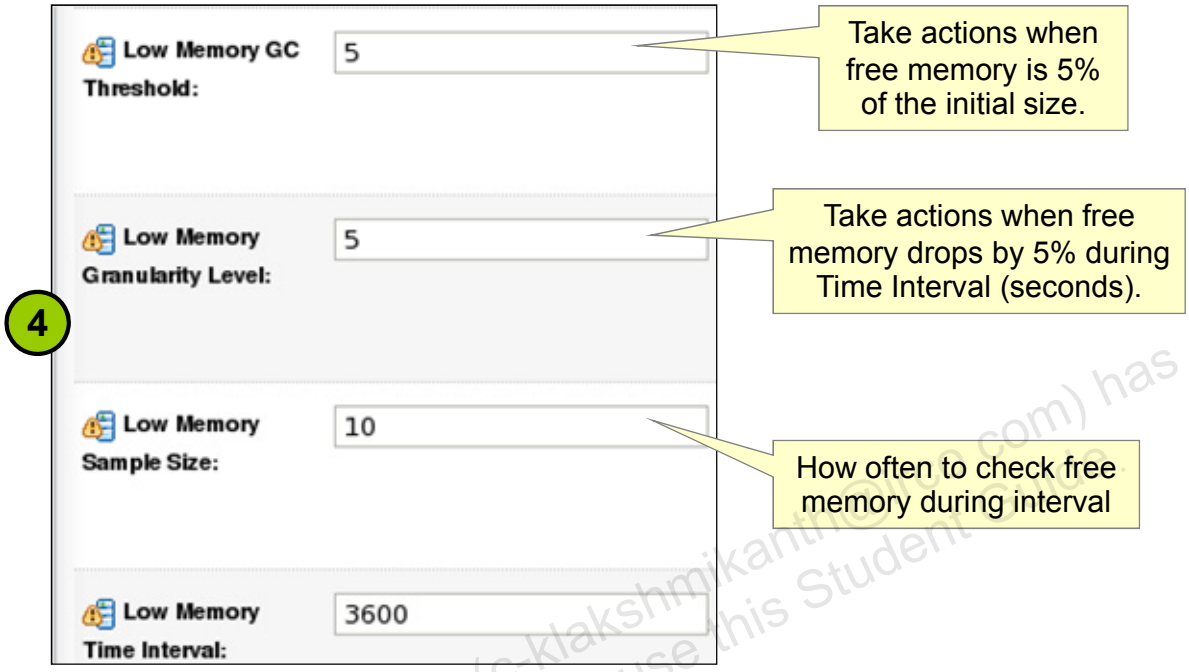
- When WLS detects a low memory condition, it:
  - Changes its health state attribute to Warning
  - Generates a log message
- Optionally create WLDF or SNMP notifications for this attribute

WLS Low Memory Detection

You can automatically log low memory conditions observed by the server. WebLogic Server detects low memory by sampling the available free memory a set number of times during a time interval. At the end of each interval, an average of the free memory is recorded and compared to the average obtained at the next interval. If the average drops by a user-configured amount after any sample interval, the server logs a low memory warning message in the log file and sets the server health state to WARNING.

1. Select a server in your domain.
2. Confirm that the Configuration tab is selected.
3. Click the Tuning tab.

Configuring Low Memory Detection

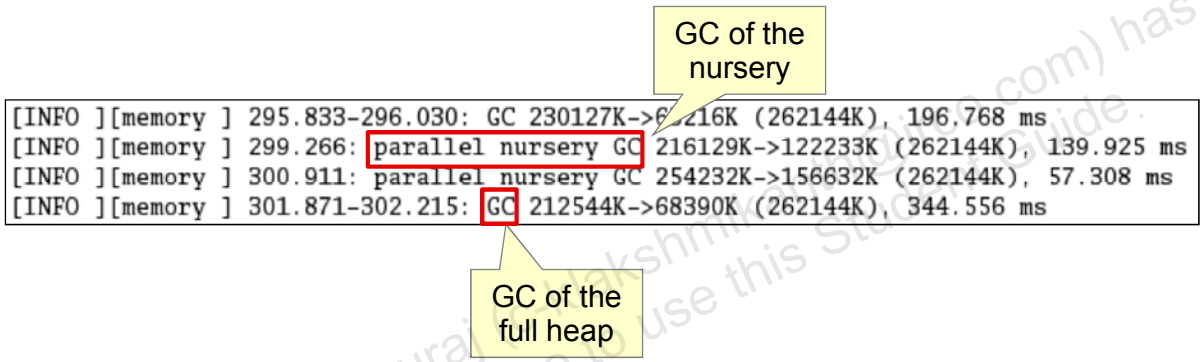


Configuring Low Memory Detection

- **Low Memory GC Threshold:** The threshold level (in percent) that this server uses for logging low memory conditions and changing the server health state to Warning. For example, if you specify 5, the server logs a low memory warning in the log file and changes the server health state to Warning after the average free memory reaches 5% of the initial free memory measured at the server's boot time.
- **Low Memory Granularity Level:** The granularity level (in percent) that this server uses for logging low memory conditions and changing the server health state to Warning. For example, if you specify 5 and the average free memory drops by 5% or more over two measured intervals, the server logs a low memory warning in the log file and changes the server health state to Warning.
- **Low Memory Time Interval:** The amount of time (in seconds) that defines the interval over which this server determines average free memory values. By default, the server obtains an average free memory value every 3600 seconds. This interval is not used if the JRockit VM is used, because the memory samples are collected immediately after a VM-scheduled garbage collection. Taking memory samples after a garbage collection gives a more accurate average value of the free memory.

JRockit Command-Line Heap Monitoring

Argument	Description
`-Xverbose:memory`	Prints statistics for each GC including the amount of heap recovered and time elapsed
`-XgcPause`	Prints a message for each GC
`-XgcReport`	Prints a comprehensive GC report after the program completes



JRockit Command-Line Heap Monitoring

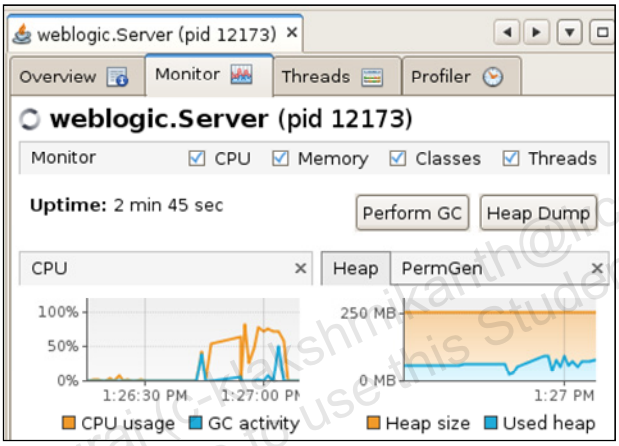
The `-Xverbose` command-line option enables verbose outputs from the Oracle JRockit JVM. You can use these verbose outputs for monitoring, tuning and diagnostics. Most verbose modules are also available in several log levels. The `-Xverbose:memory` (or `-Xverbose:gc`) module provides basic information on garbage collection events. The overhead for this module is very low, which means that you can enable it even in a production environment. Each garbage collection output starts with a timestamp, which is the time in seconds since the JVM started. At the end of each line you can also see the total heap size within parenthesis, and the total garbage collection duration for each garbage collection. Note that the garbage collection duration may consist of both pauses and concurrent garbage collection phases.

The `-XgcPause` option prints out the pause times caused by the garbage collector during a run. As pauses are encountered, they will print a report to your screen.

The `-XgcReport` option generates an end-of-run report that shows garbage collection statistics. You can use this report to determine whether you are using the most effective garbage collector for your application. The report divides the statistics into “young collections” and “old collections”, and for each of the types the following information is printed: Number of collections, Total promoted, Max promoted, Total GC time, Mean GC time, and Maximum GC pauses.

Sun JVM Heap Monitoring

- The `-verbose:gc` JVM argument prints statistics for each GC including the amount of heap recovered and time elapsed.
- The JDK also includes the JVisualVM tool, which connects to a JVM and monitors its heap and garbage collections.



Sun JVM Heap Monitoring

JVisualVM is a tool that provides a visual interface for viewing detailed information about Java applications while they are running on a JVM, and for troubleshooting and profiling these applications. JVisualVM combines all of the functionality available in the JDK's command-line diagnostic tools, such as `jstat`, `jinfo`, `jmap`, and `jstack`.

JVisualVM is able to access and display data about applications running on remote hosts. After connecting to a remote host, JVisualVM can display general data about the application's runtime environment and can monitor memory heap and thread activity. JVisualVM can retrieve monitoring information on remote applications but it cannot profile remote applications.

When you start JVisualVM, the Applications panel is visible in the left side of the main window. The Applications panel uses a tree structure to enable you to quickly view the applications that are running on local and connected remote JVM software instances. It also displays core dumps and application snapshots. For most nodes in the Applications panel, you can view additional information and perform actions by right-clicking a node and selecting an item from the popup menu.

JRockit Mission Control (JRMC)

- JRMC is a suite of graphical, low-impact JVM monitoring and diagnostic tools:
  - Management Console
  - Flight Recorder
  - Memory Leak Detector
- Check http://www.oracle.com/technetwork/middleware/jrockit/downloads/index.html for the most recent release.
- As a best practice, run JRMC on a separate machine (not on the machine running the server JVM).

JRockit Mission Control (JRMC)

The suite of tools included in Oracle JRockit Mission Control are designed to monitor, manage, profile, and gain insight into problems occurring in your Java application without requiring the performance overhead normally associated with these types of tools. You can launch the JRockit Memory Leak Detector, the JRockit Runtime Analyzer, and the JRockit Management Console from within the JRockit Mission Control Client.

To view real-time behavior of your application and of JRockit JVM, you can connect to an instance of the JVM and view real-time information through the JRockit Management Console. Typical data that you can view is thread usage, CPU usage, and memory usage. All graphs are configurable and you can add your own attributes and redefine their respective labels. In the Management Console, you can also create rules that trigger on certain events (for example, sending an email if the CPU load reaches 90%).

The JRockit Memory Leak Detector is a tool for discovering and finding the cause for memory leaks in a Java application. The JRockit Memory Leak Detector's trend analyzer discovers slow leaks, it shows detailed heap statistics (including referring types and instances to leaking objects), allocation sites, and it provides a quick drill down to the cause of the memory leak. The Memory Leak Detector uses advanced graphical presentation techniques to make it easier to navigate and understand the sometimes complex information.

Management Console Features

- JRMC tools provide a main menu, with each menu option displaying a collection of tabs.

Menu	Tab Descriptions
General	View a summary of current CPU and memory usage.
MBeans	• View all available JVM attributes. • Define notifications based on JVM attribute values.
Runtime	• View CPU usage, general JVM statistics, and Java system properties. • View more detailed memory and GC statistics. • View thread statistics and stack traces.
Advanced	• Profile JVM over time and collect statistics for selected methods and exceptions. • Run `jrcmd` commands (`print_threads`, ...).

Management Console Features

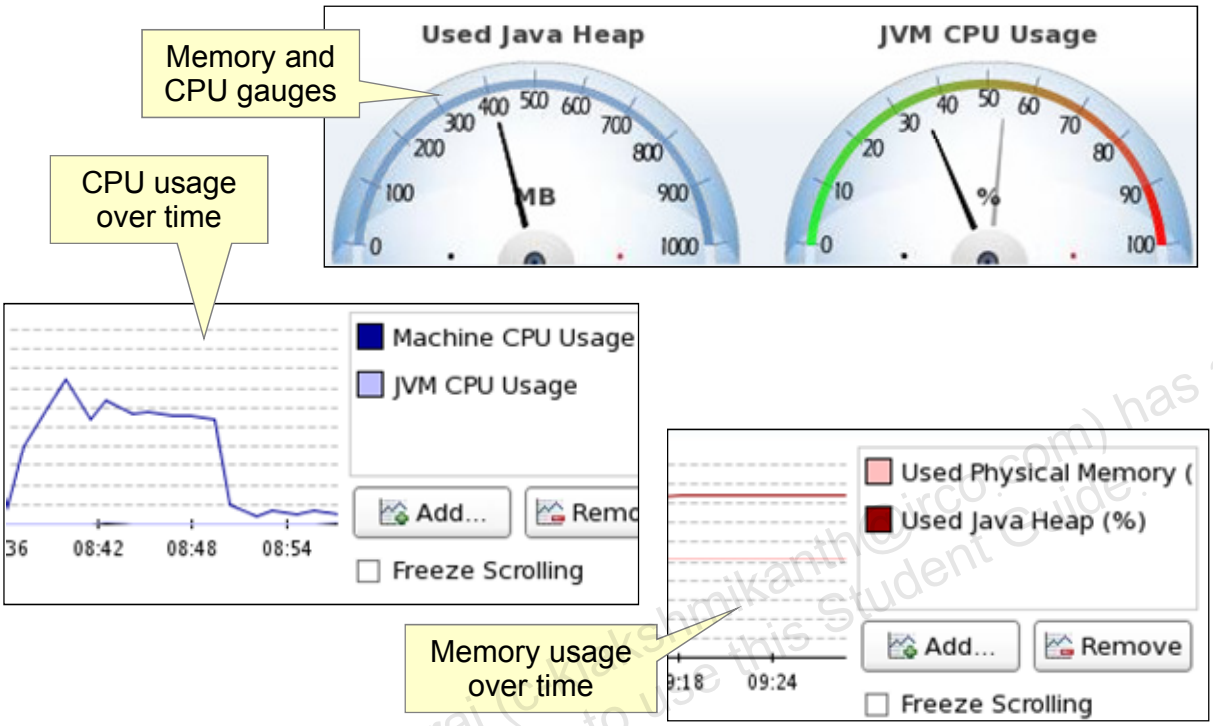
Each tab in the Management Console focuses on a specific set of JVM MBeans, although the MBean Browser tab lets you browse and monitor the raw data for all available MBeans.

The Runtime > Memory tab allows you to monitor how efficiently your application is using the memory available to it. This tab focuses on heap usage, memory usage, and garbage collection schemes. The information provided on this tab can greatly assist you in determining whether you have configured the JRockit JVM to provide optimal application performance.

The Runtime > System tab provides such information as the average processor load over time and as a percentage of the overall CPU load. It also lists all system properties loaded with the JVM.

The Advanced > Exception Count tab supports a type of profiling that counts the number of exceptions of a certain type, providing information that is helpful when you are troubleshooting your Java application.

Management Console Heap Monitoring



Management Console Heap Monitoring

The Overview tab found under the General menu option allows you to monitor processor behavior and memory statistics during system run time. The Overview tab is helpful in analyzing a system's general health as it can reveal behavior that might indicate bottlenecks or other sources of poor system performance.

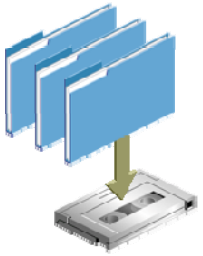
Graphs show the performance of one or more MBean attributes over time. You can add or remove attributes to any graph in the Management Console. The values are plotted on the X-axis (vertical axis) and are usually specified in percentages or raw numbers. The time element is plotted on the Y-axis (horizontal axis). Time is displayed in increments of seconds, minutes, or hours. Each graph has a legend that uses a color patch to identify the attribute being plotted.

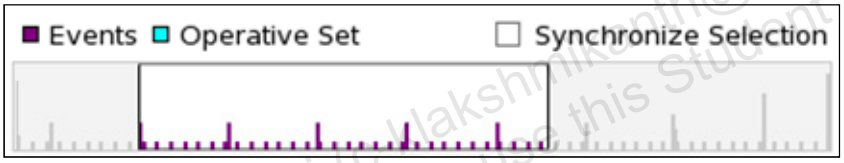
The Threads tab found under the Runtime menu option allows you to monitor thread activity for a running application. This tab contains both a graph that plots thread usage by an application over time and a sortable list of all live threads used by the application. It will also display thread stack traces.

Some Management Console tabs display data using tables. You can add and remove elements in a Management Console table and you can also resize the widths of the elements and the way they are sorted when viewing.

Flight Recorder Overview

- The Management Console analyzes current JVM performance.
- The Flight Recorder:
  - Profiles the JVM over time for performance
  - Can be started/stopped from JRMC or the command line
  - Generates a binary recording that you can view in JRMC
  - Can also be used as a "black box" to help troubleshoot the conditions leading to some JVM or application problem





Flight Recorder Overview

The JRockit Flight Recorder (JFR) is a performance monitoring and profiling tool that makes diagnostics information always available, even in the wake of catastrophic failure, such as a system crash. At its most basic, JFR is a rotating buffer of diagnostics and profiling data that is always available, on demand. You might consider it a sort “time machine” that enables you to go back in time to gather diagnostics data leading up to an event. The data stored in the rotating buffer includes JVM and application events.

Use JRMC to view the JVM’s recordings, current recording settings, and runtime parameters on a series of tabs that aggregate performance data into logical, task-based groups. The data on these tabs is presented by way of an assortment of dials, chart, and tables. At the top of each tab is a sliding window, called the Range Navigator, with which you can expand or narrow the range of reporting. For example, if you see a group of events clustered around a specific time period, you can adjust the Range Navigator to include just those events, with the resulting data for just those events appearing on the tab components.

Flight Recorder Features

Menu	Tab Descriptions
General	• View a summary of your memory usage, CPU usage, JVM arguments, and host environment.
Memory	• View a summary of memory usage, garbage collection statistics, and strategy changes. • Visualize memory usage over time and view detailed GC statistics. • Visualize memory distribution in terms of object size within the JVM. • View a list of threads that allocated the most memory. • View a list of types that required the most memory.
Code	• View a list of packages and classes that required the most CPU time. • View a list of methods that required the most CPU time along with who invoked them. • View a list of exceptions that occurred.
CPU/Threads	• Visualize CPU usage and thread creation over time. • View a list of threads that required the most CPU time. • Visualize the events that caused threads to wait (GC, file I/O, network I/O, locks, and so on).

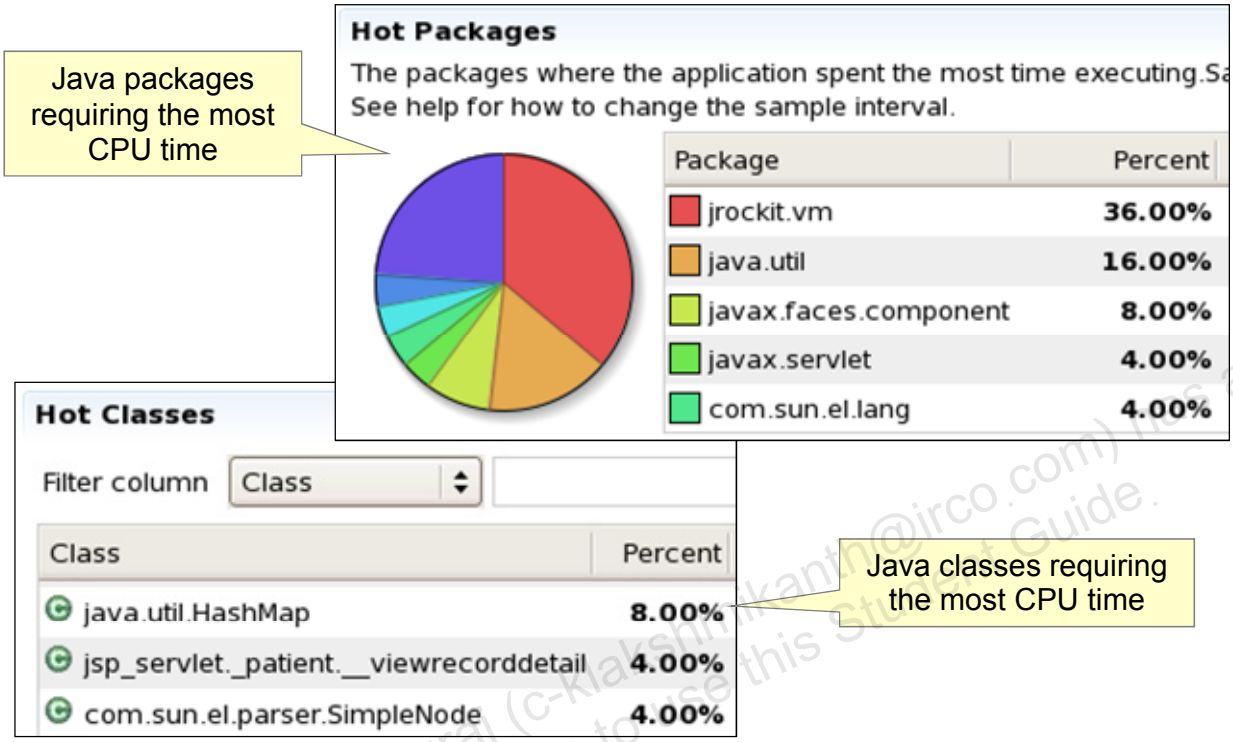
Flight Recorder Features

The General tab group of the Flight Recorder displays high-level information about the selected recording. It displays CPU and heap usage along with general information about your JVM.

The Memory tab group consists of several tabs. The Overview tab includes a graph depicting heap and physical memory usage, along with a list of JVM heap settings and general statistics. The GC General tab provides some more detailed garbage collection statistics for each generation. The Allocation tab includes several tables that describe the types of objects that were created during the recording and the threads that allocated them. The Heap Contents tab is used to help assess how efficiently the VM was able to defragment its memory. At the beginning and end of a recording session, snapshots are taken of the most common types and classes of object types that occupy the Java heap, that is, the types which instances in total occupy the most memory. The results are shown on the Object Statistics tab. Abnormal results in the object statistics might help you detect the existence of a memory leak in your application.

The CPU/Threads tab group helps you diagnose threading problems, such as those that appear to use more CPU than expected, block excessively, or cause deadlocks.

Flight Recorder Code Analysis



Flight Recorder Code Analysis

The Code tab group provides an analysis of the code that ran during the recording. The Overview tab lists the Java packages and classes that required the most CPU time. The Hot Methods is similar but drills down to the method level. The Exceptions tab lists any uncaught exceptions as well as the code that triggered them.

Section Summary

In this section, you should have learned how to:

- Describe how memory is organized using the JRockit and Sun Hotspot JVMs
- Tune the heap and garbage collection settings used by a WLS instance
- Monitor JVM performance using the WLS console or with JVM command-line arguments
- List some capabilities of JRockit Mission Control

Practice 17-1:
Tune a Server JVM

This practice covers the following topics:

- Recording a Grinder script using the TCPProxy
- Starting a Grinder agent from The Grinder console
- Modifying server JVM settings
- Monitoring heap usage with JRockit Mission Control

Road Map

- Tuning Concepts
- JVM Tuning
- WLS Tuning
  - Chunk Settings
  - Logging Considerations
  - JSP Compilation
  - Cluster Considerations
- Work Managers

Production Mode

When a domain runs in production mode:

- JRockit JVM is used by default
- Certain developer-oriented and often resource-consuming features are disabled
- The default JDBC data source maximum capacity is increased to 25 connections

Memory Chunk Settings

- Chunks are:
  - Temporary buffers used by sockets for network I/O
  - Pooled and reused to prevent allocation costs
- The default chunk size is 4KB and the default pool size is 2048 chunks.
- Increase the chunk size for servers that consistently process large payloads.

Override default server chunk settings:

```
startWebLogic.sh -Dweblogic.Chunksize=3072
                 -Dweblogic.utils.io.chunkpoolsize=3072
```


Memory Chunk Settings

A chunk is a unit of memory that the WebLogic Server network layer, both on the client and server side, uses to read data from and write data to sockets. To reduce memory allocation costs, a server instance maintains a pool of these chunks. For applications that handle large amounts of data per request, increasing the value on both the client and server sides can boost performance. The default chunk size is about 4 K.

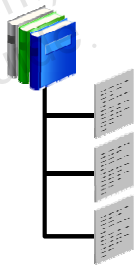
The primary situation in which the chunk size may need to be increased is when request sizes are large. It should be set to values that are multiples of the network's maximum transfer unit (MTU), after subtracting from the value any Ethernet or TCP header sizes.

The chunk pool size may need to be increased if the server starts to allocate and discard chunks in steady state. To determine whether the value needs to be increased, monitor the CPU profile or use a memory/heap profiler for call stacks invoking the constructor `weblogic.utils.io.Chunk`. The default value is 2048 chunks.

The `weblogic.PartitionSize` variable sets the number of pool partitions used (default is 4 KB). The chunk pool can be a source of significant lock contention as each request to access to the pool must be synchronized. Partitioning the thread pool spreads the potential for contention over more than one partition.

Logging Considerations

- Ensure that applications perform minimal logging.
- Set log severity thresholds to the highest acceptable levels.
- Disable standard output and HTTP logging.
- Disable any diagnostic logs associated with WLDF, JMS, JDBC, and so on.
- Use log filters to further refine which messages are written to disk or sent to the administration server.



Logging Considerations

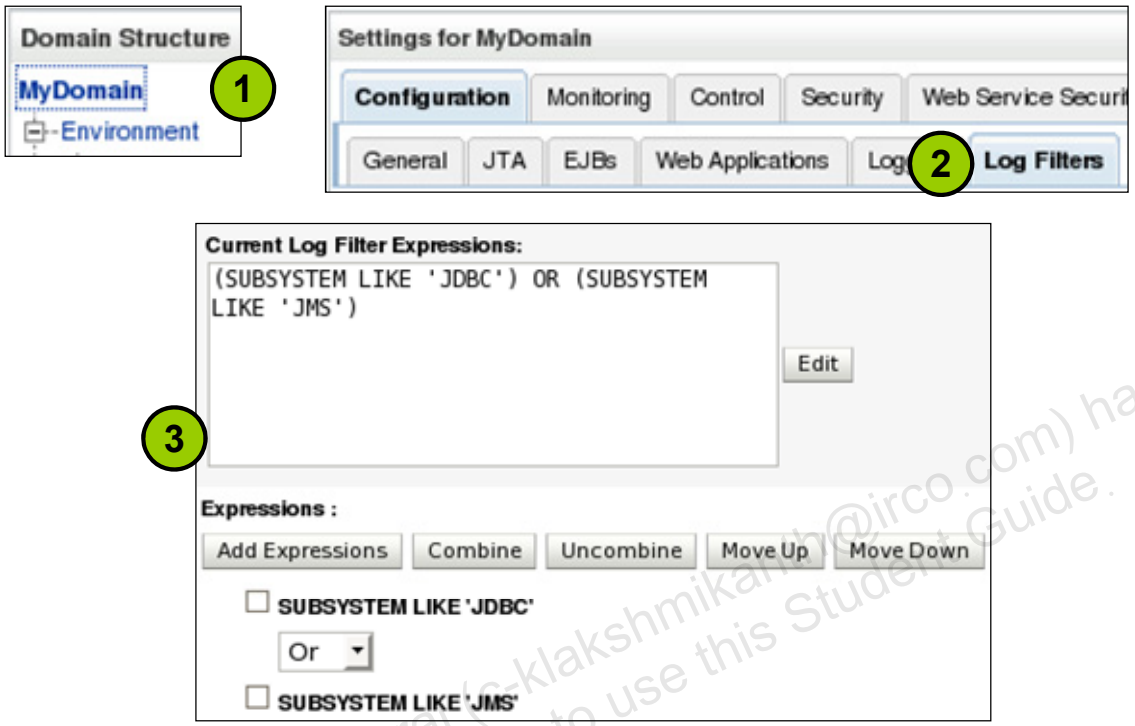
Under each server's advanced logging configuration attributes, locate the following:

- Log File Severity Level: The minimum severity of log messages going to the server log file. By default all messages go to the log file.
- Standard Out Severity Level: The minimum severity of log messages going to the standard out. Messages with a lower severity than the specified value will not be published to standard out.
- Domain Log Severity Level: The minimum severity of log messages going to the domain log from this server's log broadcaster. Messages with a lower severity than the specified value will not be published to the domain log.

By default, the HTTP subsystem keeps a log of all HTTP transactions in a text file. The default location and rotation policy for HTTP access logs is the same as the server log. Disable the HTTP access log from the Logging > HTTP tab. You can also tune HTTP logging for individual Web applications by using the XML `weblogic.xml` descriptor.

Finally, confirm that each JMS destination (topic or queue) does not enable message logging. Select a destination and click Configuration > Logging.

Log Filters Review



Log Filters Review

Log filters provide control over the log messages that get published. A filter uses custom logic to evaluate the log message content, which you use to accept or reject a log message (for example, to filter out messages of a certain severity level, from a particular subsystem, or according to specified criteria). Only the log messages that satisfy the filter criteria get published. You can create separate filters for the messages that each server instance writes to its server log file, standard out, memory buffer, or broadcasts to the domain-wide message log.

1. Select the name of the active domain in the Domain Structure panel.
2. Click Configuration > Log Filters. Click New. Enter a value to identify the filter in the Name field, and click Finish.
3. Edit the new filter. Enter the criteria for qualifying messages. You can click Edit to type or paste in an expression directly, using the WLDF Query Language syntax, or you can click Add Expression to construct an expression using a set of dialogs.

JavaServer Page (JSP) Review

JSP files:

- Are components of JavaEE Web applications
- Are HTML documents that are interwoven with Java code, dynamic expressions, and custom tags
- Provide a dynamic response that is based on the client's request
- Must be translated and compiled into Java classes before being run by the server

JavaServer Page (JSP) Review

JSPs enable you to separate the dynamic content of a Web page from its presentation. This technology caters to two different types of developers: HTML developers, who are responsible for the graphical design of the page, and Java developers, who handle the development of software to create the dynamic content.

Because JSPs are part of the Java EE standard, you can deploy JSPs on a variety of platforms, including WebLogic Server. In addition, third-party vendors and application developers can provide JavaBean components and define custom JSP tags that can be referenced from a JSP page to provide dynamic content.

WebLogic Server handles JSP requests in the following sequence:

1. A browser requests a page with a .jsp file extension from WebLogic Server.
2. Using the JSP compiler, WebLogic Server converts the JSP into a Servlet class. The JSP file is compiled only when the page is first requested, or when the JSP file has been changed. Otherwise, the previously compiled JSP Servlet class is reused, making subsequent responses much quicker.
3. The generated Servlet class is invoked to handle the browser request and respond with dynamic content, such as a Web page.

Precompiling JSP Files

- By default, JSP files are compiled the first time they are accessed by clients.
- To avoid this initial overhead, use WLS tools to precompile JSP files before deployment.
- Place compiled Java classes in the application's `WEB-INF/classes` folder.

Precompile JSPs from the command line:

```
java weblogic.appc hrWeb.war
```


Precompiling JSP Files

In addition to compiling plain Java source files, the WebLogic `weblogic.appc` compiler generates classes from JSP source files and also validates XML descriptors. The –verbose argument is option and can be useful for testing or troubleshooting purposes.

An alternative approach is to enable precompilation during application deployment, using the Web application's `weblogic.xml` descriptor. However, with this approach, JSPs are compiled every time the application is restarted.

JSP and Class Reloading

- During development, WLS automatically checks for changes to your JSP and class files and reloads them without requiring an explicit redeployment.
- In production mode, these features are disabled ("-1") by default.
- Verify that Web applications do not override the defaults.

weblogic.xml:

```
<jsp-descriptor>
   <page-check-seconds>-1</page-check-seconds>
</jsp-descriptor>
...
<container-descriptor>
   <servlet-reload-check-secs>-1</servlet-reload-check-secs>
   <resource-reload-check-secs>-1</resource-reload-check-secs>
</container-descriptor>
```


JSP and Class Reloading

WebLogic Server has the ability to automatically check a deployed Web application to see if newer versions of JSPs or other Java classes are available. If discovered, these later versions are loaded into memory instead. However, this feature is disabled by default. If these features were being used during development, be sure to disable them in production environments where these types of "on the fly" changes are rare. This will improve server performance. Either remove these elements from application deployment descriptors or set their values to "-1."

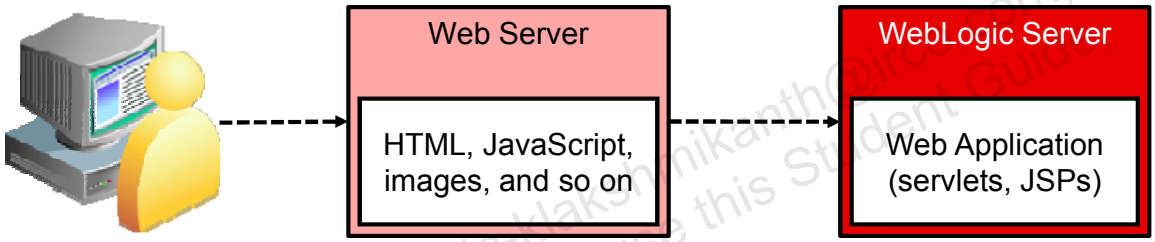
`page-check-seconds`: Sets the interval, in seconds, at which WebLogic Server checks to see whether JSP files have changed and need recompiling. Dependencies are also checked and recursively reloaded if changed.

`servlet-reload-check-secs`: Defines whether a WebLogic Server will check to see whether a servlet has been modified, and if it has been modified, reloads it.

`resource-reload-check-secs`: Used to perform metadata caching for cached resources that are found in the resource path in the Web application scope. This parameter identifies how often WebLogic Server checks whether a resource has been modified and if so, it reloads it.

Using Web Servers for Static Content

- Front application servers and clusters with a dedicated Web tier to serve static content such as HTML and image files.
- Proxy requests for servlets and JSPs to application servers.
- Perform additional optimizations at the Web tier, such as caching.



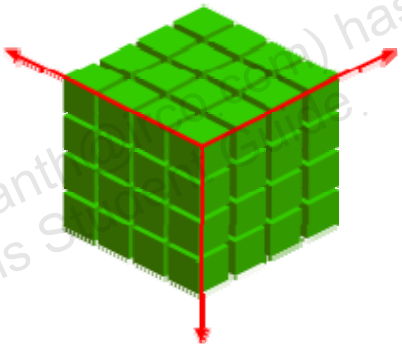
Using Web Servers for Static Content

Web servers provide a hosting environment that is optimized for delivering static content and files over HTTP. They typically support common server features such as single sign-on, access control, and high availability. It is also common for Web servers to provide an execution environment for simple applications that generate dynamic content using languages such as Perl.

WebLogic Server is in fact a Web server, because it can provide these services as well. However, WebLogic Server is optimized for processing Java application requests and for serving dynamic Web content generated from servlets and JSPs. Therefore, if your Web application includes a significant number of static file resources such as HTML documents, images and JavaScript libraries, you may achieve greater performance and scalability by offloading these static resources from WebLogic Server to one or more dedicated Web servers.

Cluster Considerations

- File and JDBC session persistence typically have a significantly larger overhead than in-memory replication.
- Because WLS replicates sessions at the attribute level, application developments should attempt to:
  - Use sessions sparingly (consider cookies)
  - Group data that often changes together into an attribute
  - Minimize the size of each attribute



Cluster Considerations

WebLogic Server offers five session persistence mechanisms that cater to the differing requirements of your application. The session persistence mechanisms are configurable at the Web application layer. Which session management strategy you choose for your application depends on real-world factors like HTTP session size, session life cycle, reliability, and session failover requirements. In terms of pure performance, in-memory session persistence is a better overall choice when compared to JDBC-based persistence for session state. However, in-memory-based session persistence requires the use of WebLogic clustering, so it is not an option in a single-server environment.

As a general rule, you should optimize your application so that WebLogic Server does as little work as possible when handling session management and persistence. The use of sessions involves a scalability trade-off in most contexts. Use sessions only for states that cannot realistically be kept on the client or if URL rewriting support is required. For example, keep simple bits of states, such as a user's name, directly in cookies. Similarly, keep frequently used values in local variables if appropriate.

Aggregate session data that changes in tandem into a single session attribute. If you use a single large attribute that contains all the session data and only 10% of that data changes, the entire attribute has to be replicated, causing unnecessary network overhead.

Session Persistence Cache Size

For file or JDBC persistence, use the cache size to control how many sessions are kept in server memory before being swapped to the persistent store.

weblogic.xml:

```
...
<session-descriptor>
   <persistent-store-type>file</persistent-store-type>
   <cache-size>2056</cache-size>
   ...
</session-descriptor>
```


Session Persistence Cache Size

You can configure the number of sessions that are held in memory for an application by using several parameters in the <session-descriptor> element of the weblogic.xml deployment descriptor file.

The `<cache-size>` parameter limits the number of cached sessions that can be active in memory at any one time. It is only applicable for file and JDBC persistence. If you expect high volumes of simultaneous active sessions, you do not want these sessions to soak up the RAM of your server because this may cause performance problems swapping to and from virtual memory. When the cache is full, the least recently used sessions are stored in the persistent store and recalled automatically when required. By default, the number of cached sessions is 1028. To turn off caching, set this to 0.

The `<invalidation-interval-secs>` parameter sets the time, in seconds, that WebLogic Server waits between doing house-cleaning checks for timed-out and invalid sessions, and deleting the old sessions to free up memory. Use this element to tune WebLogic Server for best performance on high-traffic sites. The default value is 60 seconds.

Section Summary

In this section, you should have learned how to:

- List some simple techniques that may increase server or application performance
- Describe the use of chunks for network I/O
- Precompile JSP files before application deployment

Practice 17-2:
Tune Server Performance

This practice covers the following topics:

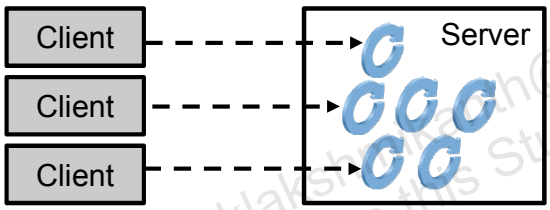
- Monitoring data source performance
- Tuning data source connection pool settings
- Tune server log settings

Road Map

- Tuning Concepts
- JVM Tuning
- WLS Tuning
- Work Managers
  - Request Classes
  - Constraints
  - Assigning to Applications
  - Stuck Threads

WebLogic Server Threads

- Like all server software, WLS processes concurrent requests by assigning each to a thread from an available pool.
- WLS 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.

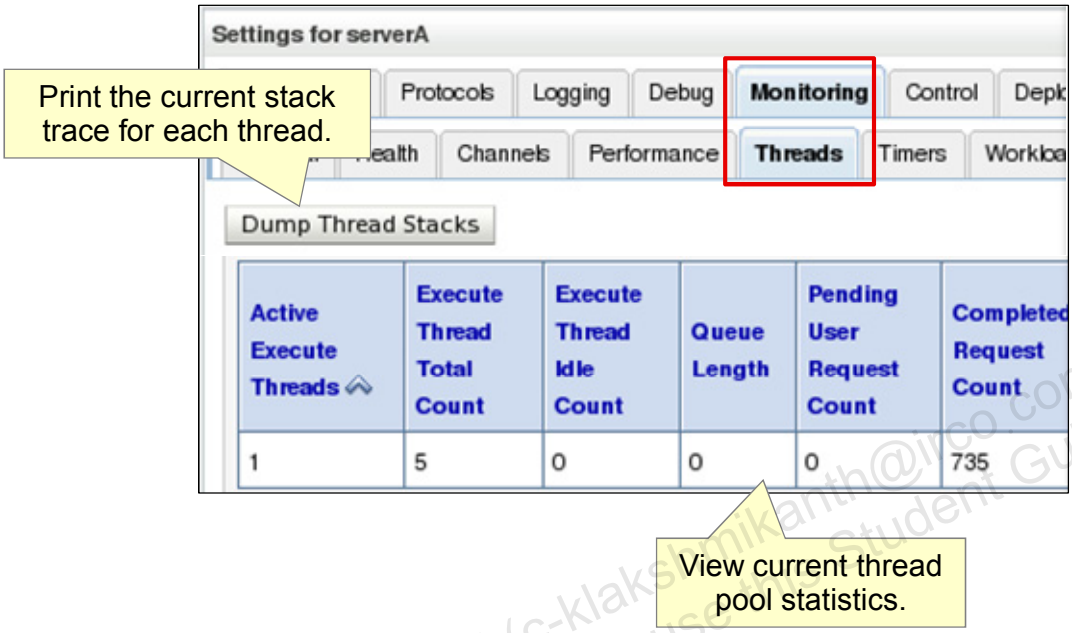


WebLogic Server Threads

WebLogic Server uses a single thread pool, in which all types of work are executed. WebLogic Server 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 are entering and leaving 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



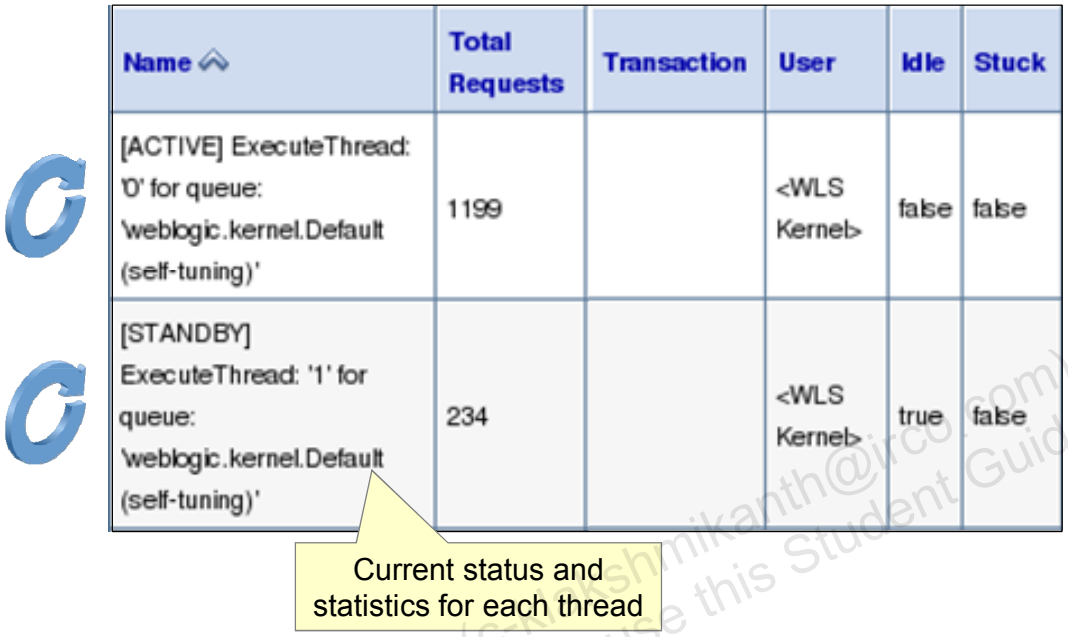
Monitoring a Server Thread Pool

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 on 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



Monitoring Server Threads

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 that the execute thread is currently working on behalf of.
- **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 running for a long time are likely deadlocked, 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 its state to Failed after a specified number of threads become stuck
  - Can automatically shut itself down if in the Failed state

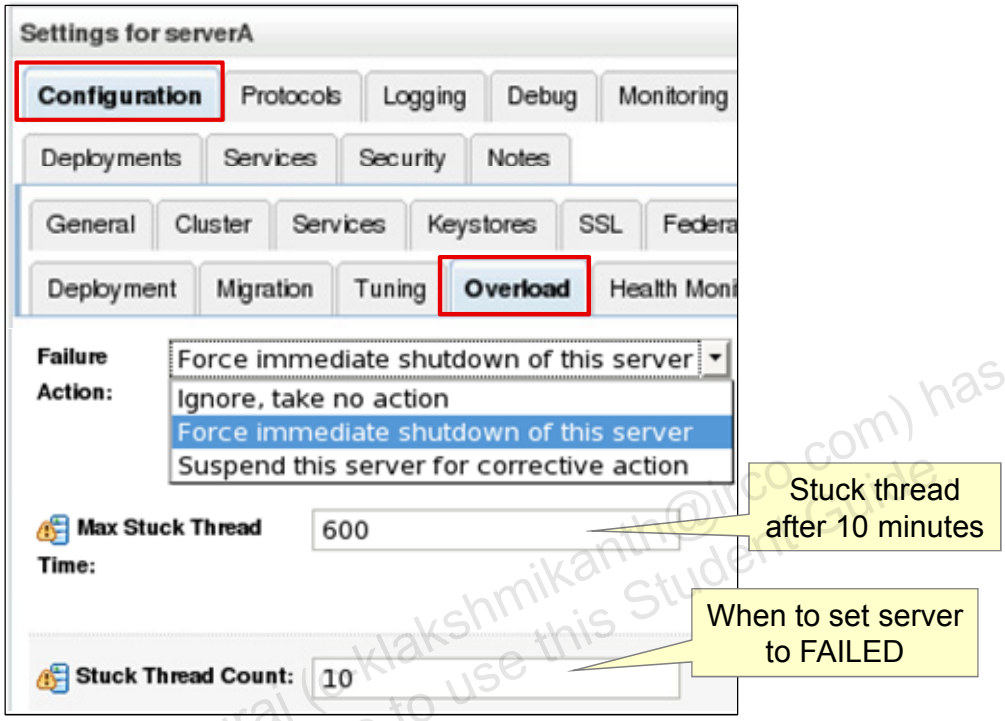
Stuck Thread Handling

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 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 admin mode. All Work Managers belonging to the application are shut down, and behave as described above.

Configuring Stuck Thread Handling



Stuck Thread Handling (continued)

Select a server and locate its Configuration > Overload tab. The following fields relate to stuck thread handling:

- **Shared Capacity For Work Managers**: 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.
- **Enable Failure Trigger:** Mark a server as failed if threads are stuck.
- **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.

Stuck Thread Handling (continued)

- 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.

weblogic-application.xml:

```
...
<application-admin-mode-trigger>
   <max-stuck-thread-time>300</max-stuck-thread-time>
   <stuck-thread-count>5</stuck-thread-count>
</application-admin-mode-trigger>
```


Application Stuck Thread Handling

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, that a thread must execute an application request before being considered stuck.

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 WLS using a work manager.
- If not assigned a work manager, applications use the default one.
- To override the SLA characteristics of the default work manager, create one named “default.”

Work Managers

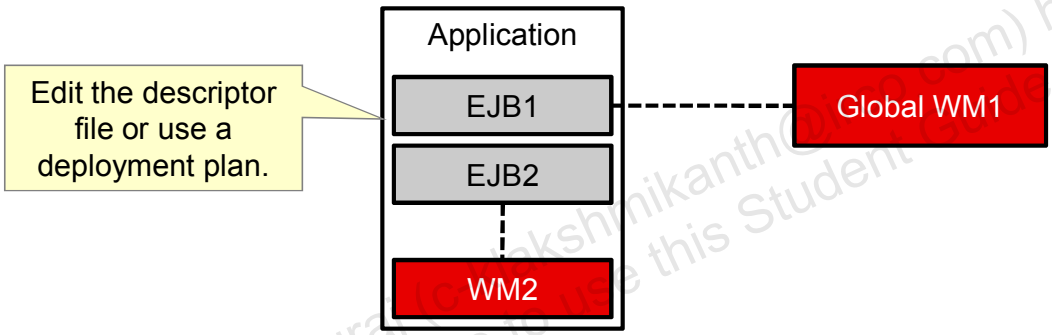
WebLogic Server allows you to configure how your application prioritizes the execution of its work. Based on rules that you define and by monitoring actual runtime performance, WebLogic Server can optimize the performance of your application and maintain service level agreements (SLA). You tune the thread utilization of a server instance by defining rules and constraints for your application by defining a work manger and applying it either globally to WebLogic Server 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 Server 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 Server’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 allows you to control the default thread-handling behavior of WebLogic Server.

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.



Work Manager Scope

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 WLS 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.

Work Manager Architecture

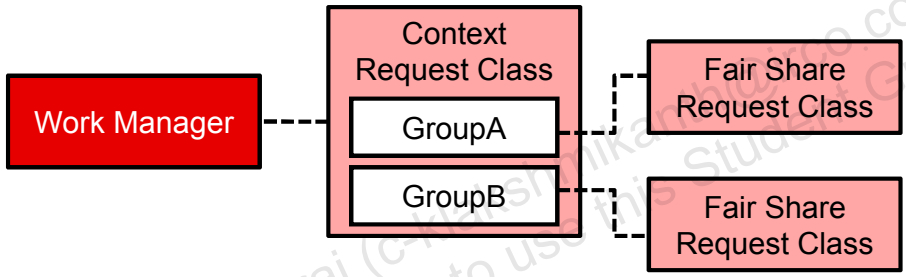
A request class expresses a scheduling guideline that WebLogic Server uses to allocate threads to requests. Request classes help ensure that high priority work is scheduled before less important work, even if the high priority work is submitted after the lower priority work. WebLogic Server takes into account how long it takes for requests to each module to complete.

A constraint defines minimum and maximum numbers of threads allocated to execute requests and the total number of requests that can be queued or executing before WebLogic Server begins rejecting requests.

In response to stuck threads, you can define an error handing policy that shuts down the work manager, moves the application into admin mode, or marks the entire server instance as failed.

Request Class Types

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)



Request Class Types

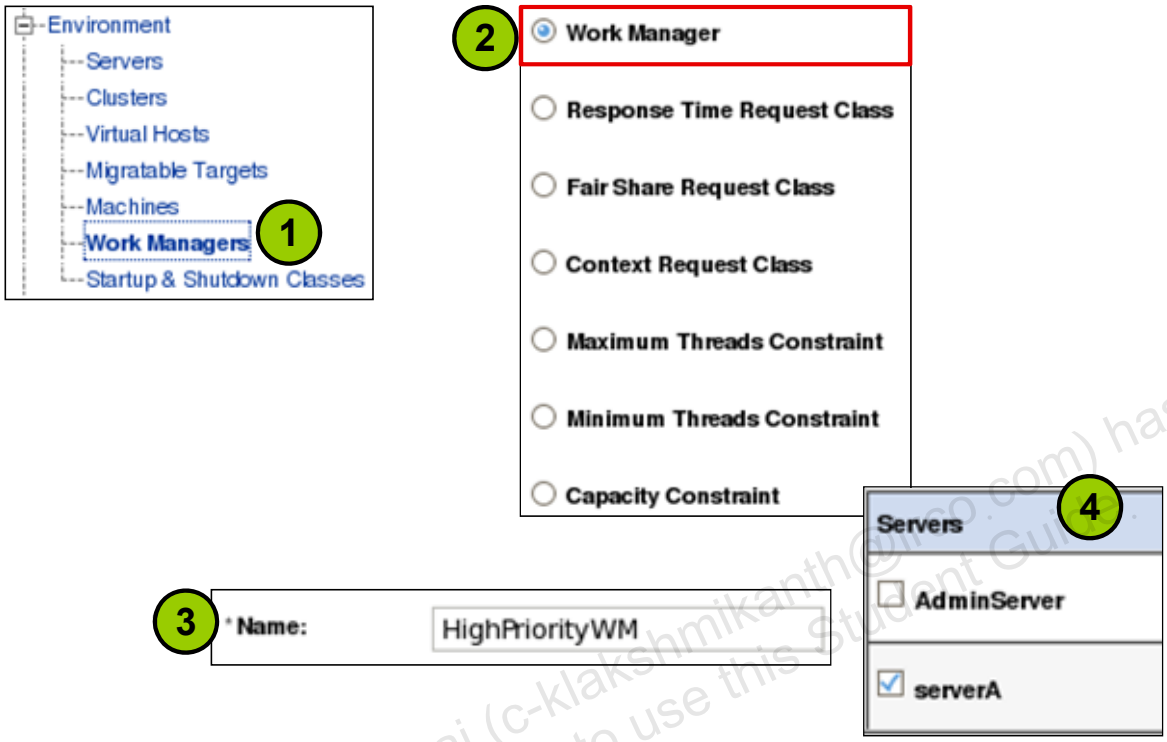
There are multiple types of request classes, each of which expresses a scheduling guideline in different terms. A work manager may specify only one request class.

A fair share request class specifies the average thread-use time required to process requests. For example, assume that WebLogic Server 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 Server 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 Server 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

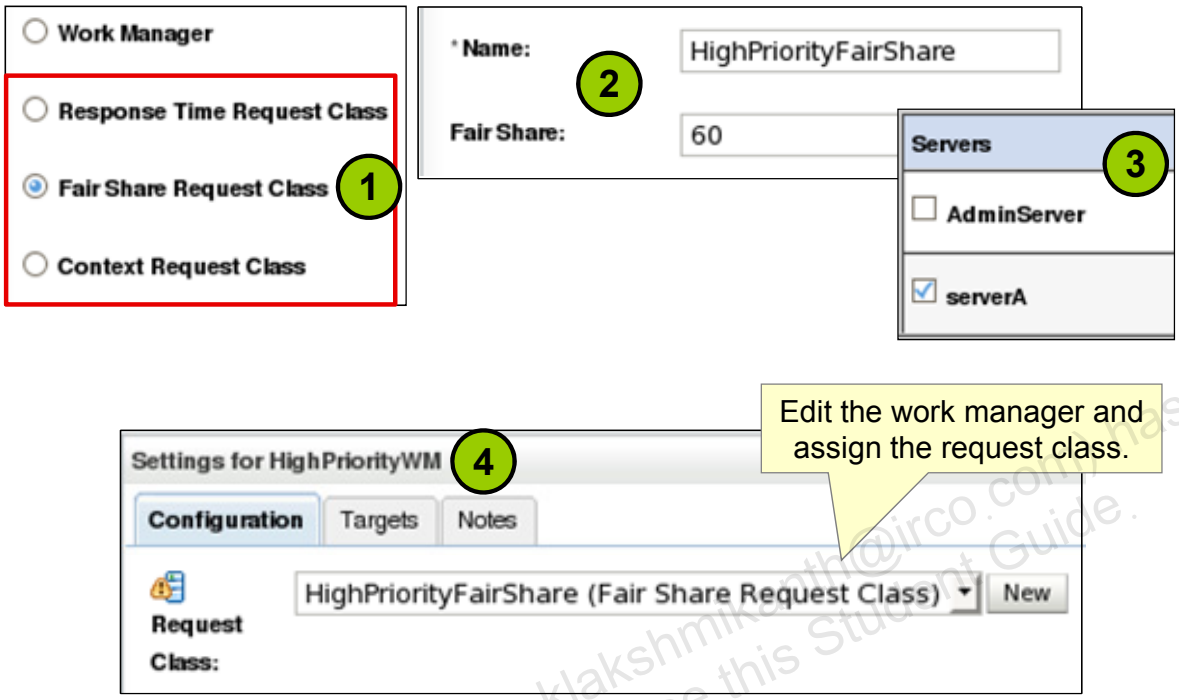


Creating a Work Manager

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. Enter a Name for the new work manager, and click Next.
4. In the Available Targets list, select server instances or clusters on which you will deploy applications that reference the work manager. Then click Finish.

Creating a Request Class



ORACLE

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 server instances or clusters on which you will deploy applications that reference this request class. Then click Finish.
4. Edit you existing work manager. Select your new request class by using the Request Class field, and click Save.

Constraint Types

Constraint Type	Description
Maximum Threads	• The maximum number of concurrent threads that can be used to service requests • Can be a static value or 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

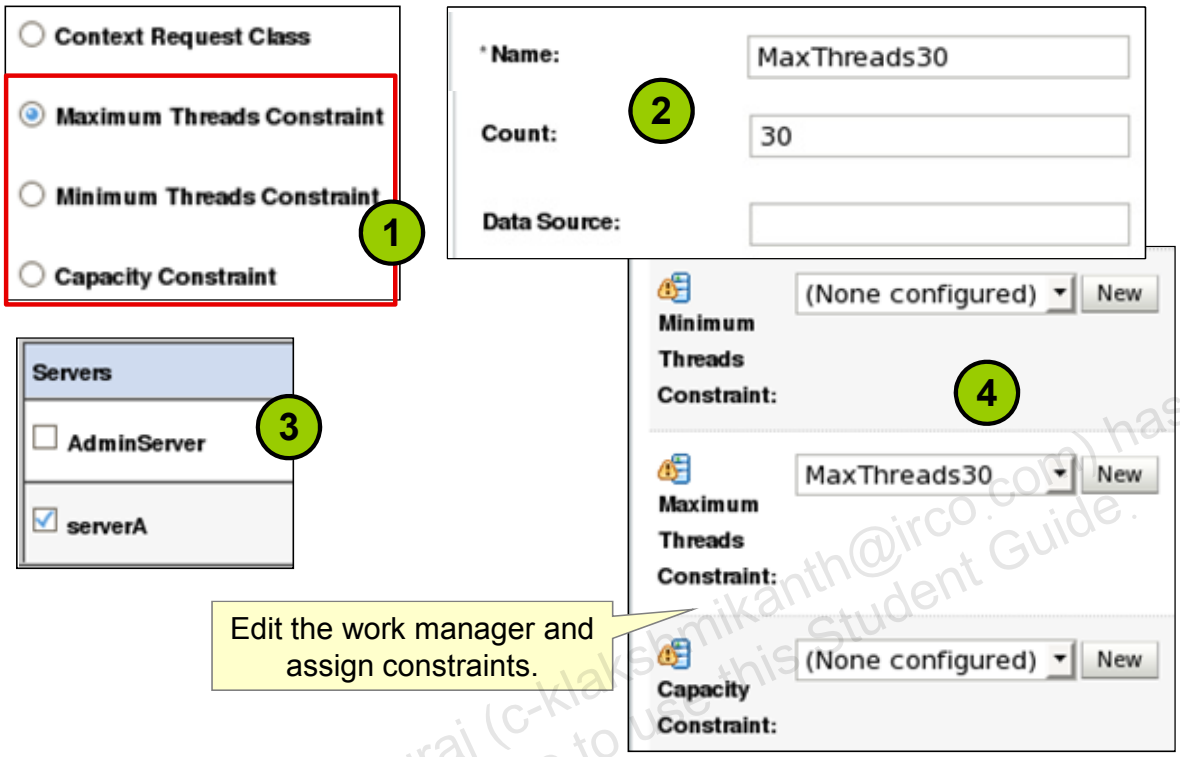
Constraint Types

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 Server 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



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 upon 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')
```

ORACLE

Work Manager WLST Example

Refer to the following MBeans:

- 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)
```


Work Managers and Stuck Threads

The `WorkManagerShutdownTriggerMBean` 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 WM to a Web application or module by using `weblogic.xml`:

```
...
<wl-dispatch-policy>HighPriorityWM</wl-dispatch-policy>
```

Assign a WM 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>
```


Assigning Work Managers to Applications

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.

Section Summary

In this section, you should have learned how to:

- Describe how WLS handles concurrent client requests
- Monitor thread pool size and usage
- Prioritize application processing using work managers
- Tune work managers using request classes and constraints
- Update an application to use a work manager

Quiz

Name three components of the The Grinder tool.

a. Proxy
b. Agent
c. Profiler
d. Chunk
e. Console

Answer: a, b, e

Quiz

What term refers to garbage collection algorithms that are based on object age?

a. Heaping
b. Timed
c. Generational
d. Precompiled

Answer: c

Quiz

Which of the following is NOT a type of work manager request class?

a. Context
b. Fair Share
c. Response Time
d. Cluster

Answer: d

Lesson Summary

In this lesson, you should have learned how to:

- Define basic performance terms such as scalability, benchmarking, and stress test
- Use The Grinder to record and run simple load tests
- List some basic command-line arguments for tuning the JRockit and Sun Hotspot JVMs
- Monitor a server's heap and garbage collectors by using various command-line and graphical tools
- Tune WLS networking and logging.
- Create work managers to implement SLAs for applications

Practice 17-3:
Tune Performance 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

18

Monitoring and Diagnostics Essentials

ORACLE

Objectives

After completing this lesson, you should be able to:

- Describe WebLogic Server's JMX architecture
- Monitor WLS runtime MBeans
- List the capabilities of WLDF
- Configure diagnostic collectors and monitors
- Monitor WLS by using SNMP
- Enable debugging for WLS subsystems

Road Map

- Monitoring Concepts
  - Monitoring and Diagnostic Tools
  - JMX Runtime Hierarchy
  - WLS Monitoring Review
  - FMW Control Overview
  - Guardian Overview
- WLDF Fundamentals
- WLS SNMP
- WLS Debugging

Why Monitoring Tools?

- Use monitoring and diagnostic software to:
  - Periodically assess the current health and/or performance of the server
  - Analyze historical health and/or performance metrics
  - Automatically receive notifications (email and so on) when the server's workload and/or performance is not at expected levels
  - Troubleshoot server or application errors
- Monitoring can occur at various levels:
  - Server or JVM
  - Server subsystem (JMS, JTA, and so on)
  - Application or resource

Why Monitoring Tools?

Monitoring the health of your Oracle Fusion Middleware environment and ensuring that it performs optimally is an important task for the administrator. You can monitor the status of Oracle WebLogic Server domains, servers, their underlying Java Virtual Machines (JVMs), server components, and applications.

Oracle Monitoring Tools

Oracle provides several tools to support your monitoring and/or troubleshooting requirements, including:

- WLS Administration Console
- WebLogic Server Scripting Tool (WLST)
- WebLogic Diagnostics Framework (WLDF)
- WLS SNMP Agent
- JRockit Mission Control
- Oracle Fusion Middleware (FMW) Control
- Oracle Grid Control
- Oracle Guardian

Oracle Monitoring Tools

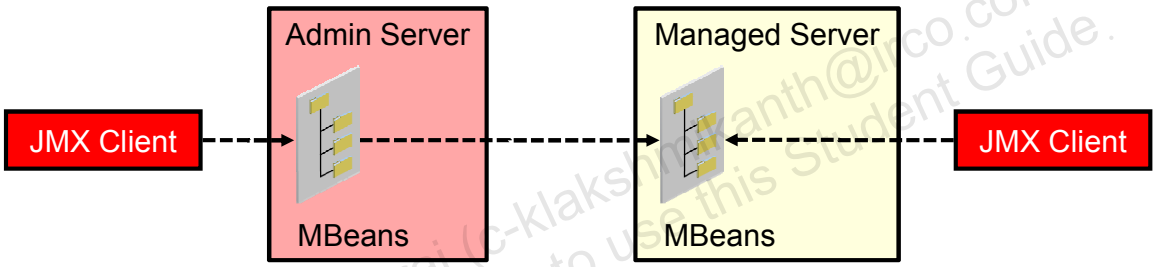
You can monitor the status and performance of Oracle WebLogic Server domains, servers, Java components, and applications by using Oracle WebLogic Server Administration Console. You can also use various command-line tools like WLST to perform the same tasks. To periodically monitor and publish this information to some external system or repository, you can use WebLogic Server's diagnostic framework and/or its SNMP agent.

You can view the overall status of the Oracle Fusion Middleware (FMW) environment from the home page of a farm by using the Fusion Middleware Control. The farm's home page lists FMW components, including Web servers and WebLogic Server domains. The FMW Control also provides an application deployment summary.

Oracle Guardian is a diagnostics tool that you can use to periodically scan WebLogic Server domains for common configuration mistakes or security vulnerabilities. Using Oracle Guardian can be thought of as similar to using a virus scanner against an operating system.

Java Management Extension (JMX) Review

- All WLS configuration, management, and monitoring is accomplished via JMX, which provides a standard Java API for accessing management “objects” or MBeans.
- MBeans:
  - Are organized hierarchically
  - Can be accessed from remote systems
  - On the administration server act as proxies to MBeans on managed servers



Java Management Extension (JMX) Review

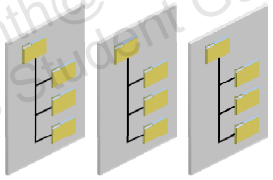
A managed bean (MBean) is a Java object that provides a Java Management Extensions (JMX) interface. JMX is the JavaEE solution for monitoring and managing resources on a network. Like SNMP and other management standards, JMX is a public specification and many vendors of commonly used monitoring products support it. WebLogic Server provides a set of MBeans that you can use to configure, monitor, and manage WebLogic Server resources through JMX.

Each server in the domain has its own copy of the domain’s configuration documents (which consist of a `config.xml` file and subsidiary files). During a server’s startup cycle, it contacts the Administration Server to update its configuration files with any changes that occurred while it was shut down. Then it instantiates configuration MBeans to represent the data in the configuration documents.

MBeans, like any Java objects, consist of attributes, which can be simple types such as strings or integers, or they can be other MBeans. In other words, MBeans can contain other MBeans and define a hierarchy of management objects. MBeans provide standard methods for managing these relationships, which are typically of the form createXYZ, destroyXYZ, or lookupXYZ, where XYZ is the type of child MBean. For example, the DomainMBean includes an operation named lookupServers(String name), which takes as a parameter the name that was used to create a server instance.

WLS MBean Hierarchies

- Every WebLogic Server hosts several different MBean containers or “servers.”
- The *Server Configuration* hierarchy lets you view the current configuration settings for resources.
- The *Server Runtime* hierarchy:
  - Lets you monitor resource statistics
  - Provides historical data since last server restart
- Administration servers also include an *Edit Configuration* hierarchy to create, update, and delete configuration MBeans for all servers in the domain.

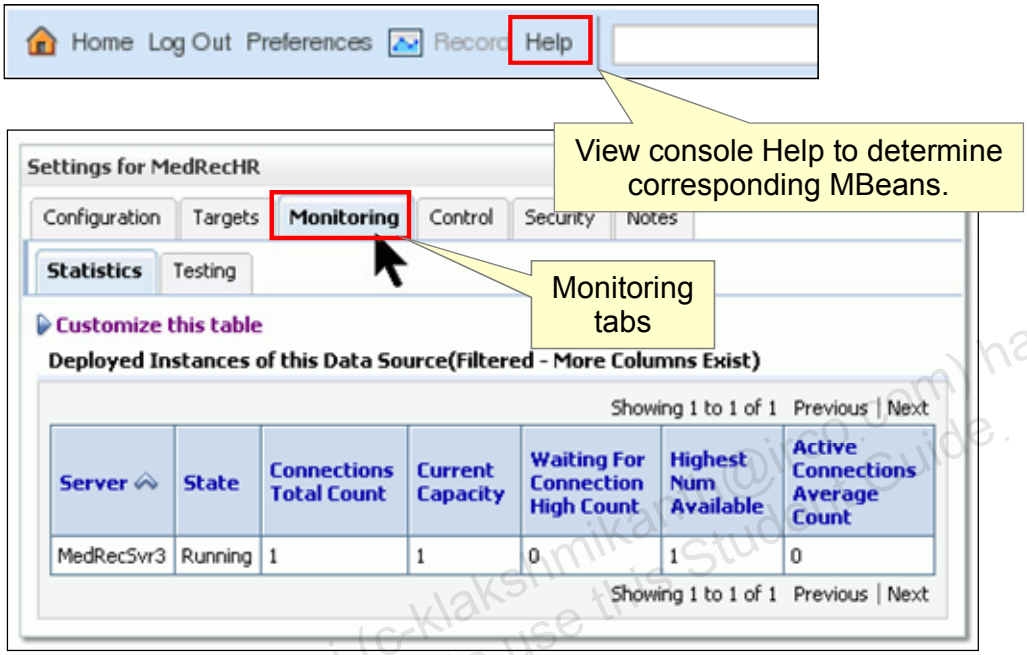


WLS MBean Hierarchies

All WebLogic Server MBeans can be organized into one of several general types. Runtime MBeans contain information about the runtime state of a server and its resources. They generally contain only data 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 destroyed. 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.

All edits to MBean attributes occur within the context of an edit session, and only one edit session can be active at a time within each WebLogic Server domain. Changing an MBean attribute or creating a new MBean updates the in-memory hierarchy of pending configuration MBeans. If you end your edit session before saving these changes, the unsaved changes will be discarded. When you activate your changes, WebLogic Server copies the saved, pending configuration files to all servers in the domain. Each server evaluates the changes and indicates whether it can consume them. If it can, it updates its active configuration files and in-memory hierarchy of configuration MBeans.

Monitoring with the Console



ORACLE

Monitoring with the Console

Every time that a resource, service, or application object can be monitored, a Monitoring tab is available in the console for that object. Clicking it shows you the available monitoring information for the selected object. Moreover, when the monitoring page shows information in tabular format, you may change the way that the information is displayed. To do this, click “Customize this table” and choose which columns to display and on what columns to sort the table.

One way to determine which MBean types and attributes correspond to the displayed columns is using the context-sensitive Help link found at the top of the console. For each column, the Help page provides a description and the corresponding MBean information.

You cannot monitor the activity of one domain through another domain. For example, you cannot open the administration console for domainY and try to monitor servers within domainZ.

Monitoring with WLST

Monitor a server work manager:

Access this server's runtime MBean hierarchy.

```
serverRuntime()
wm = getMBean('/WorkManagerRuntimes/weblogic.kernel.Default')
print 'Pending: ' + wm.getPendingRequests()
print 'Stuck Threads: ' + wm.getStuckThreadCount()
```

Monitor a server JVM:

```
serverRuntime()
jvm = getMBean('/JVMRuntime/server1')
print 'Free Heap: ' + jvm.getFreeHeapPercent()
print 'Current Heap: ' + jvm.getHeapSizeCurrent()
print 'GC Time: ' + jvm.getTotalGarbageCollectionTime()
```

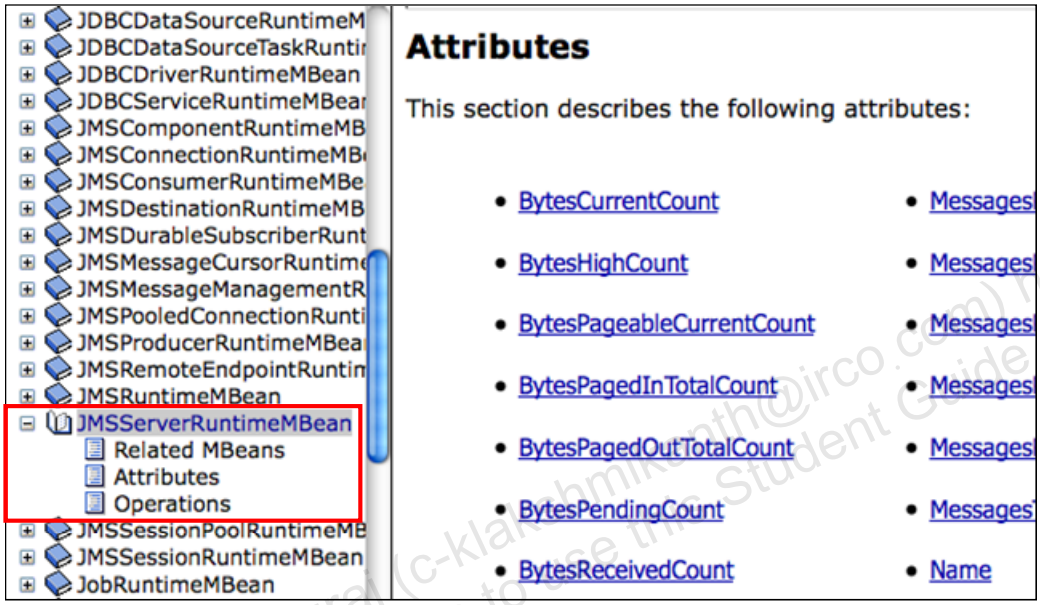

Monitoring with WLST

After connecting to a running server, the `serverRuntime()` WLST command navigates to the last MBean to which you navigated in the runtime MBean hierarchy or to the root of the hierarchy, `ServerRuntimeMBean`. This read-only hierarchy stores the runtime MBeans that represent the server to which WLST is currently connected.

You may then use all of the standard WLST browsing commands such as cd(), get(), and ls() to navigate the runtime MBean hierarchy for a server and to retrieve the desired attributes.

Runtime MBean Documentation

The available attributes and operations for all standard WLS MBeans can be found in the online documentation.



Runtime MBean Documentation

As an alternative to using WLST to interactively browse a server's runtime MBeans, you can use the online documentation, specifically the *WebLogic Server MBean Reference Guide*. Select the **Runtime MBeans** category in the tree found in the left frame. When connecting to a managed server, the root of the runtime hierarchy is ServerRuntimeMBean. From this MBean, you can then obtain references to all resources targeted to this server.

Fusion Middleware (FMW) Control Review

- Oracle Enterprise Manager's FMW Control is a Web application that can be used to:
  - Monitor the WLS domain upon which it is deployed
  - Start and stop servers
  - Browse and monitor WLS MBeans and log files
  - Deploy and monitor applications
  - Monitor and manage Web tier components
  - Monitor and manage other FMW products (SOA, Identity and Access Management, WebCenter, and so on)
- In FMW Control, a *farm* includes:
  - All the processes that make up a WLS domain
  - All Web tier processes associated with the domain

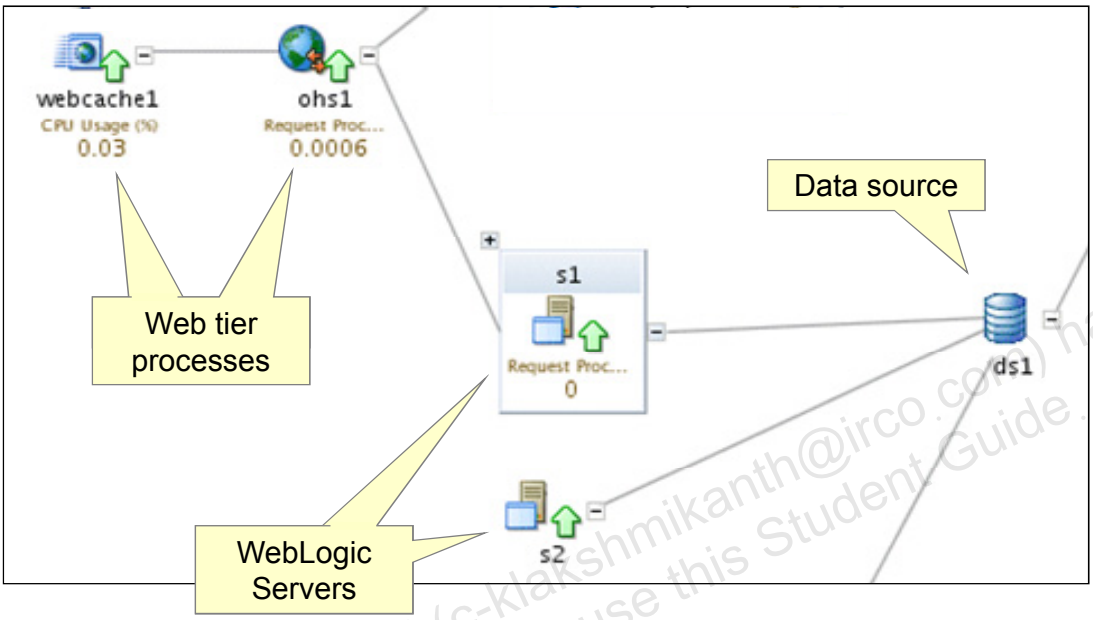
Fusion Middleware (FMW) Control Review

Fusion Middleware Control is a Web browser–based user interface that you can use to monitor and administer a farm. A farm is a collection of components managed by Fusion Middleware Control. A farm can contain an Oracle WebLogic Server domain, one Administration Server, one or more managed servers, clusters, and the Oracle Fusion Middleware components that are installed, configured, and running in the domain. These may include SOA and other product-specific components. By default, this application is accessed by using the URL http://<admin_server_host>:<port>/em. To access Fusion Middleware Control and perform tasks, you must have the appropriate role. Fusion Middleware Control uses the Oracle WebLogic Server security realm and the roles defined in that realm.

Fusion Middleware Control organizes a wide variety of performance data and administrative functions into distinct, Web-based home pages for the farm, domain, servers, components, and applications. The Fusion Middleware Control home pages make it easy to locate the most important monitoring data and the most commonly used administrative functions.

Fusion Middleware Control provides a set of MBean browsers that allow you to browse the MBeans for an Oracle WebLogic Server or for a selected application. You can also perform specific monitoring and configuration tasks from the MBean browser.

FMW Control: Viewing Farm Topology

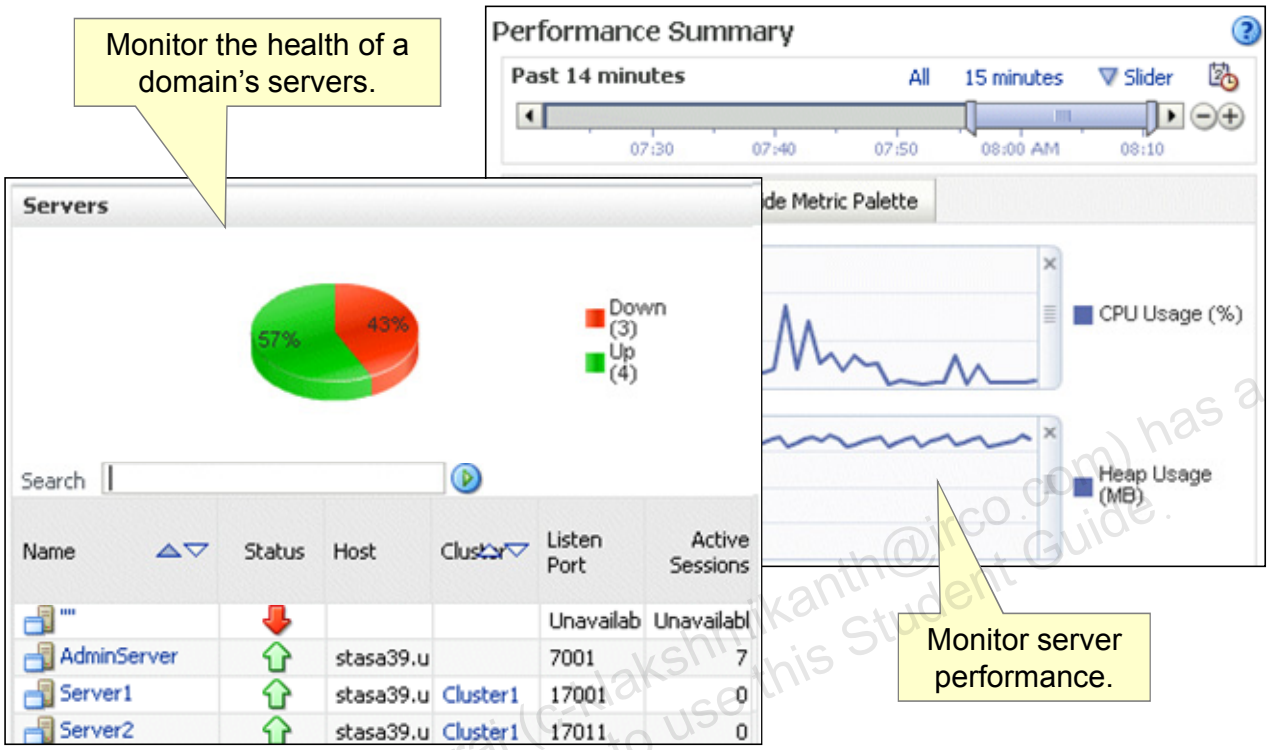


FMW Control: Viewing Farm Topology

Fusion Middleware Control provides a Topology Viewer for the farm. The Topology Viewer is a graphical representation of routing relationships across components and elements of the farm. You can easily determine how requests are routed among components. For example, you can see how requests are routed from Oracle Web Cache, to Oracle HTTP Server, to a managed server, to a data source.

From the View menu, you can save or print the image, expand or collapse all nodes, or change the orientation of the topology to be left-to-right or top-to-bottom. You can perform operations directly on the target by right-clicking, which opens a shortcut menu from which, for example, you can start or stop an Oracle WebLogic Server or view additional performance metrics.

FMW Control: Monitoring Servers



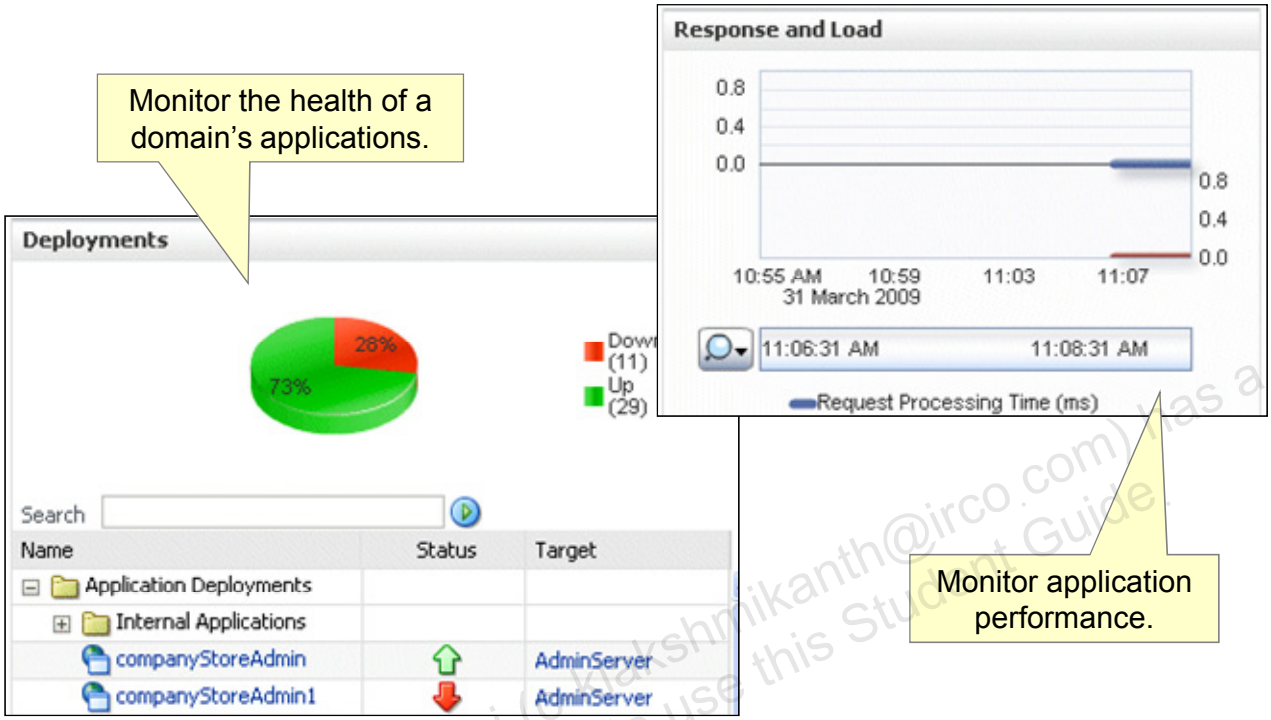
FMW Control: Monitoring Servers

You can view the status of a domain, including the servers, clusters, and deployments in the domain home page of Fusion Middleware Control. This page shows a general summary of the domain, a link to the Oracle WebLogic Server Administration Console, information about the administration server and the managed servers in the domain, and information about the clusters in the domain.

After selecting a specific server in the domain, the server home page is displayed. This page shows a general summary of the server, including its state, as well as some general statistics about the servlets, JSPs, and EJBs running on the server. This page also lets you monitor the server's response time and load.

If you encounter a problem, such as an application that is running slowly or is hanging, you can view more detailed performance information, including performance metrics for a particular target, to find out more information about the problem. A server's Performance Summary page shows server-wide performance metrics as well as metrics for individual applications deployed to the server. To see additional metrics, click Show Metric Palette and browse the available metric categories. To overlay another target application, click Overlay, and select the target. The target is added to the current charts, so that you can view the performance of more than one target at a time, comparing their performance.

FMW Control: Monitoring Deployments

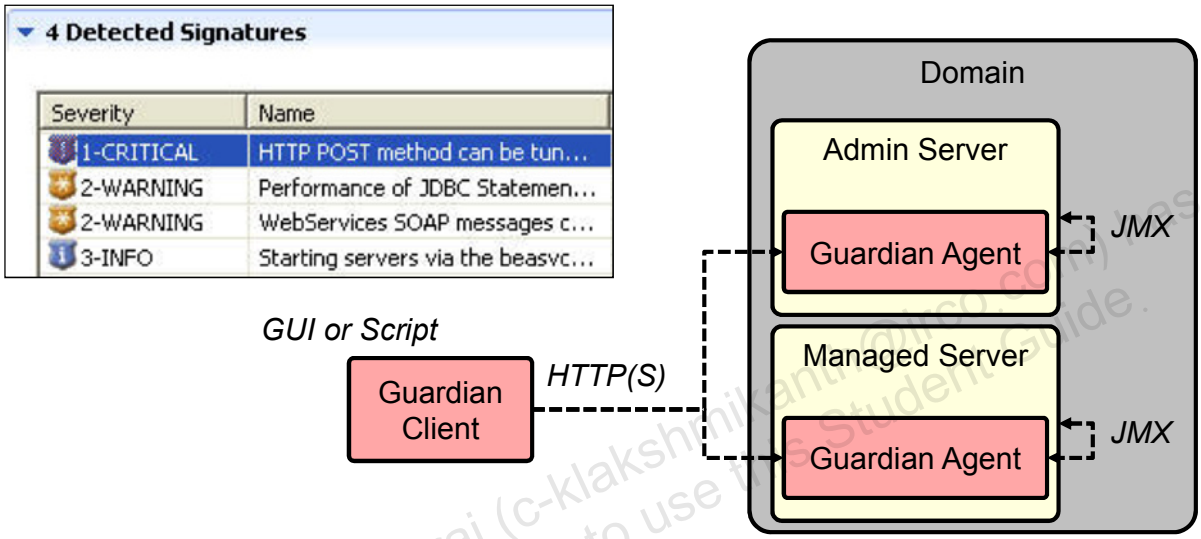


FMW Control Overview

To monitor application deployments with the FMW Control, expand Application Deployments, and then select the application to monitor. The application's home page is displayed. You can view a summary of the application's status, entry points to the application, and Web services and modules associated with the application. This page also displays the response time and load for the entire application as well as for the most requested servlets and JSPs. SOA, ADF, and WebCenter applications have additional product-specific metrics.

Guardian Overview

Oracle Guardian scans a domain's configuration for common problems and recommends solutions.



Guardian Overview

Guardian is a diagnostic tool for identifying potential problems in your environment before they occur, and provides specific instructions for resolving them. Guardian is like having the entire Oracle Customer Support Team scrutinize your domain and immediately present their findings and recommendations to you, at your convenience.

As you develop an application and migrate from development to quality assurance to production, you can run an evaluation at each stage. Guardian will help ensure that each phase of your development process is compliant with Oracle best practices. Similarly, after you install a new Oracle patch, service pack, or upgrade, or install or upgrade third-party software, you can run an evaluation to identify any new issues that may have been introduced.

Guardian can run on Windows or Linux systems that have Java Version 5 or higher installed. Guardian can evaluate any platform based on WebLogic Server version 8.1 and above, regardless of the operating system on which it is running.

Activating a domain enables it for Guardian evaluation. You can also organize multiple domains into domain groups. Then, select one or more domains, select a signature bundle, and launch an evaluation. Guardian then proceeds to evaluate the signature bundle against the specified domain(s), and generates a detailed report of potential issues and their remedies. You can then review the report and decide how to proceed.

Section Summary

In this section, you should have learned how to:

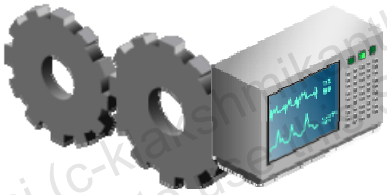
- List several tools that you can use to monitor WebLogic Server
- Describe WebLogic Server's MBean architecture
- Access runtime MBeans by using the console or WLST

Road Map

- Monitoring Concepts
- WLDF Fundamentals
  - Diagnostic Images
  - Diagnostic Archives
  - Diagnostic Modules
  - Metric Collectors
  - Watches
  - Diagnostic Monitors
  - Console Extension
- WLS SNMP
- WLS Debugging

WebLogic Diagnostics Framework (WLDF)

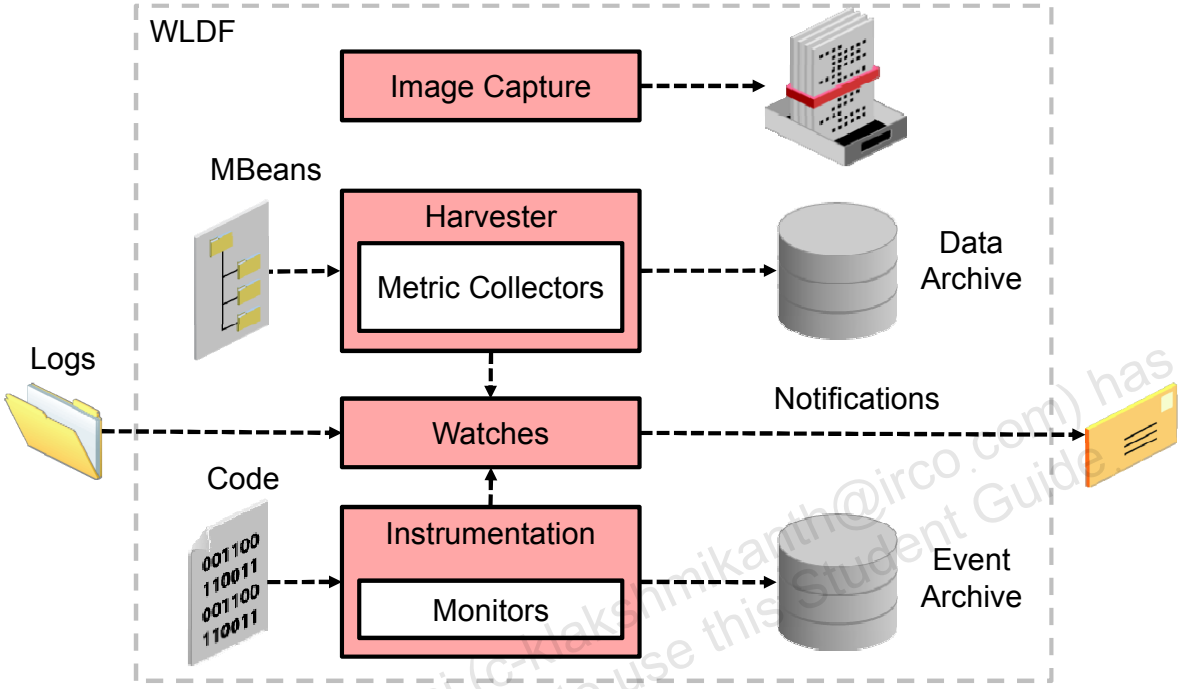
- WLDF provides a generic framework to gather and analyze WLS runtime data for monitoring and/or troubleshooting.
- Use WLDF to:
  - Capture a snapshot of key server metrics for distribution to support personnel
  - Capture metrics at specific code points in WLS or your application
  - Periodically record selected MBean attributes
  - Send notifications when attributes meet certain conditions



WebLogic Diagnostics Framework (WLDF)

WLDF consists of a number of components that work together to collect, archive, and access diagnostic information about a WebLogic Server instance and the applications it hosts.

WLDF Architecture



WLDF Architecture

Data creators generate diagnostic data that is consumed by the logger and the harvester. Those components coordinate with the archive to persist the data, and they coordinate with the watch and notification subsystem to provide automated monitoring. The data accessor interacts with the logger and the harvester to expose current diagnostic data and with the archive to present historical data. MBeans make themselves known as data providers by registering with the harvester. Collected data is then exposed to both the watch and notification system for automated monitoring and to the archive for persistence.

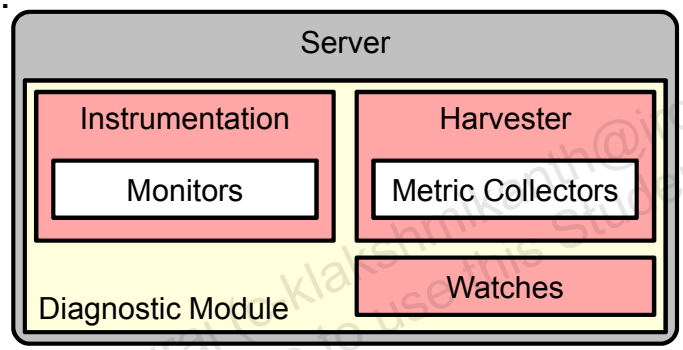
The instrumentation system creates monitors and inserts them at well-defined points in the flow of code execution within the JVM. These monitors trigger events and publish data directly to the archive. They can also take advantage of watches and notifications.

Diagnostic image capture support gathers the most common sources of the key server state used in diagnosing problems. It packages that state into a single artifact, which can be made available to support technicians.

The past state is often critical in diagnosing faults in a system. This requires that the state be captured and archived for future access, creating a historical archive. In WLDF, the archive meets this need with several persistence components. Both events and harvested metrics can be persisted and made available for historical review.

WLDF Configuration Overview

- Configure and generate diagnostic images for a server.
- Configure diagnostic archives for a server.
- Create and target a new diagnostic module.
- Add metric collectors, watches, notifications, and/or monitors to the module.
- Optionally, create application-scoped modules and monitors.



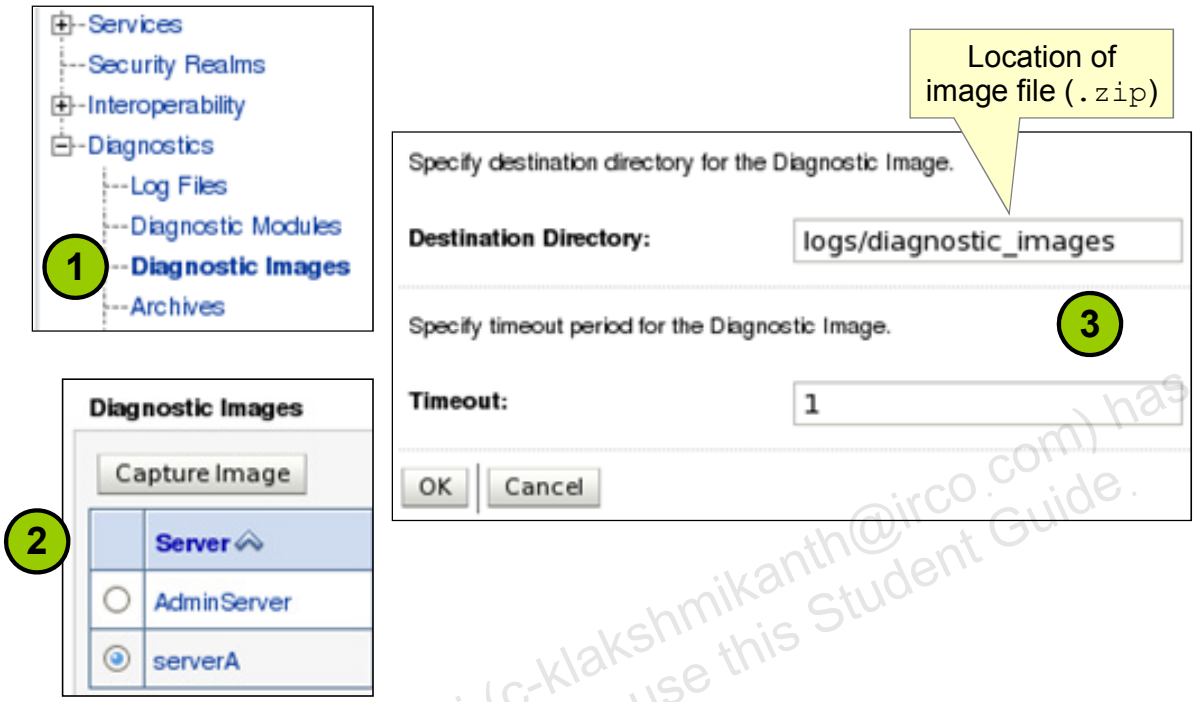
WLDF Configuration Overview

WLDF provides features for generating, gathering, analyzing, and persisting diagnostic data from WebLogic Server instances and from applications deployed to them. For server-scoped diagnostics, some WLDF features are configured as part of the configuration for a server in a domain. Other features are configured as system resource descriptors that can be targeted to servers (or clusters). For application-scoped diagnostics, diagnostic features are configured as resource descriptors for the application.

The harvester, watch, and instrumentation features are configured, packaged, and targeted as part of a diagnostic module, similar to a JMS module resource. Only instrumentation monitors can be defined in application-scoped modules, which are placed in the application's `weblogic-diagnostics.xml` file.

You create a diagnostic system module through the console or WLST. It is created as a WLDFResourceBean, and the configuration is persisted in a resource descriptor file called `<module>.xml`, where `<module>` is the name of the diagnostic module. The file is created by default in the domain's config\diagnostics directory. The file has the extension `.xml`.

Capturing a Server Diagnostic Image



Capturing a Server Diagnostic Image

You use the diagnostic image capture component of WLDF to create a diagnostic snapshot, or dump, of a server's internal runtime state at the time of the capture. This information helps support personnel analyze the cause of a server failure. You can capture an image manually using the console or WLST, or generate one automatically as part of a watch notification.

Because the diagnostic image capture is meant primarily as a post-failure analysis tool, there is little control over what information is captured. It includes the server's configuration, log cache, JVM state, work manager state, JNDI tree, and most recent harvested data. The image capture subsystem combines the data files produced by the different server subsystems into a single zip file.

If JRockit's Flight Recorder feature is enabled on your JVM, the diagnostic image will also contain the recording file, which you can view with JRockit Mission Control (JRMC). If enabled, WLS can record its own events to this JVM file, which you can also browse with JRMC.

To capture a server image using the console, perform the following:

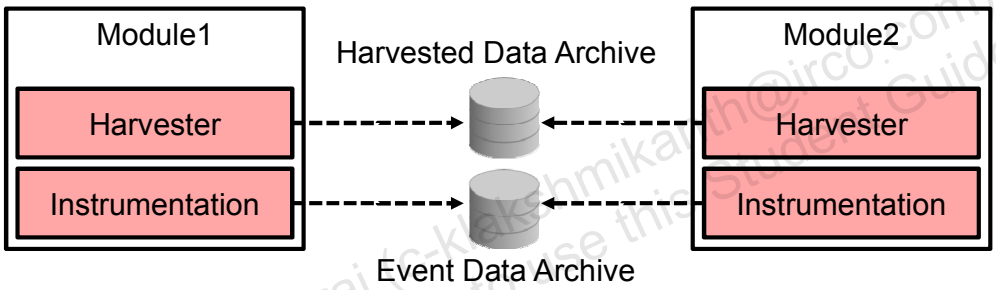
1. In the left pane, expand Diagnostics and select Diagnostic Images.
2. Select the server for which you want to capture a diagnostic image, and click Capture Image.

Capturing a Server Diagnostic Image (continued)

3. Enter a new directory in the Destination Directory field, or accept the default directory. If you change the directory for this image capture, it does not change the default directory for capturing images for this server when they are captured by other means, such as a watch notification. Then click OK.

Diagnostic Archives

- Collected MBean metrics and events are recorded to the server's diagnostic archives:
  - WLDF file store (`<server>/data/store/diagnostics`, by default)
  - WLDF JDBC store
- Recorded data can be limited using size- or age-based retirement policies.



ORACLE

Diagnostic Archives

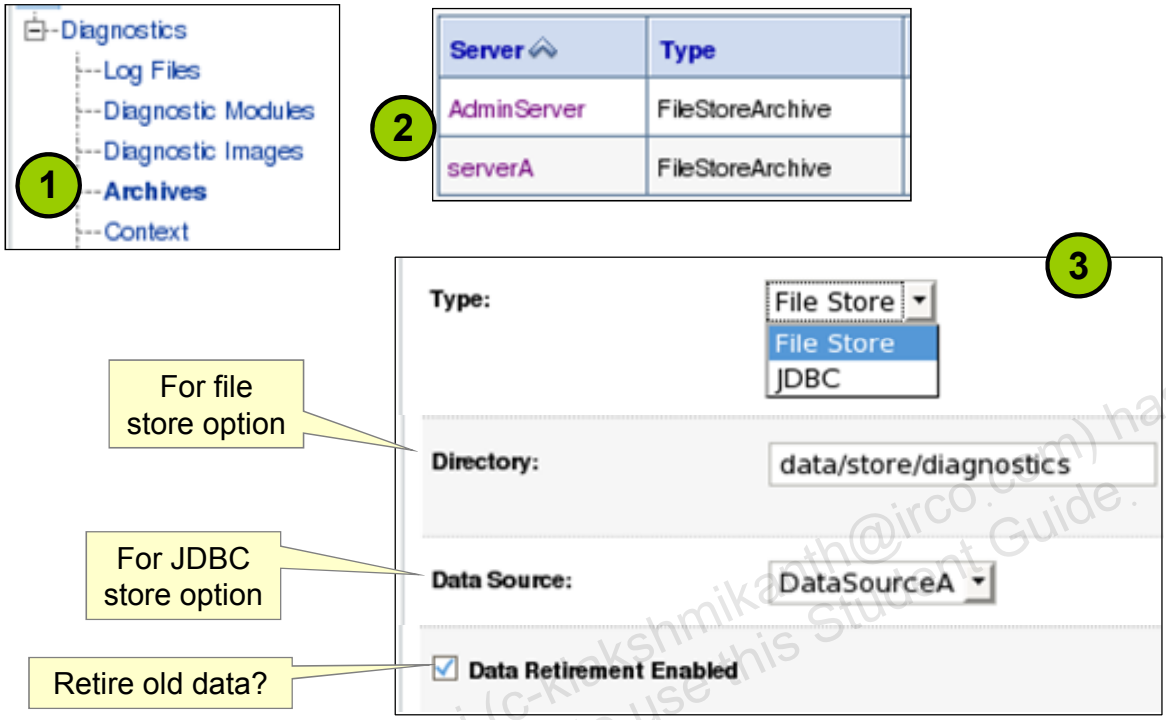
The Archive component of the WebLogic Diagnostic Framework (WLDF) captures and persists all data events, log records, and metrics collected by WLDF from server instances and applications running on them. You can access archived diagnostic data in online mode (that is, on a running server). You can also access archived data in off-line mode using the WebLogic Scripting Tool (WLST). You configure the diagnostic archive on a per-server basis.

For a file-based store, WLDF creates a file to contain the archived information. The only configuration option for a WLDF file-based archive is the directory where the file will be created and maintained. The default directory is `<domain>/servers/<server>/data/store/diagnostics`. When you save to a file-based store, WLDF uses the WebLogic Server persistent store subsystem.

To use a JDBC store, the appropriate tables must exist in a database, and JDBC must be configured to connect to that database. The `wls_events` table stores data generated from WLDF Instrumentation events. The `wls_hvst` table stores data generated from the WLDF Harvester component. Refer to the documentation for the required schema.

WLDF includes a configuration-based, data-retirement feature for periodically deleting old diagnostic data from the archives. You can configure size-based data retirement at the server level and age-based retirement at the individual archive level.

Configuring Server Diagnostic Archives

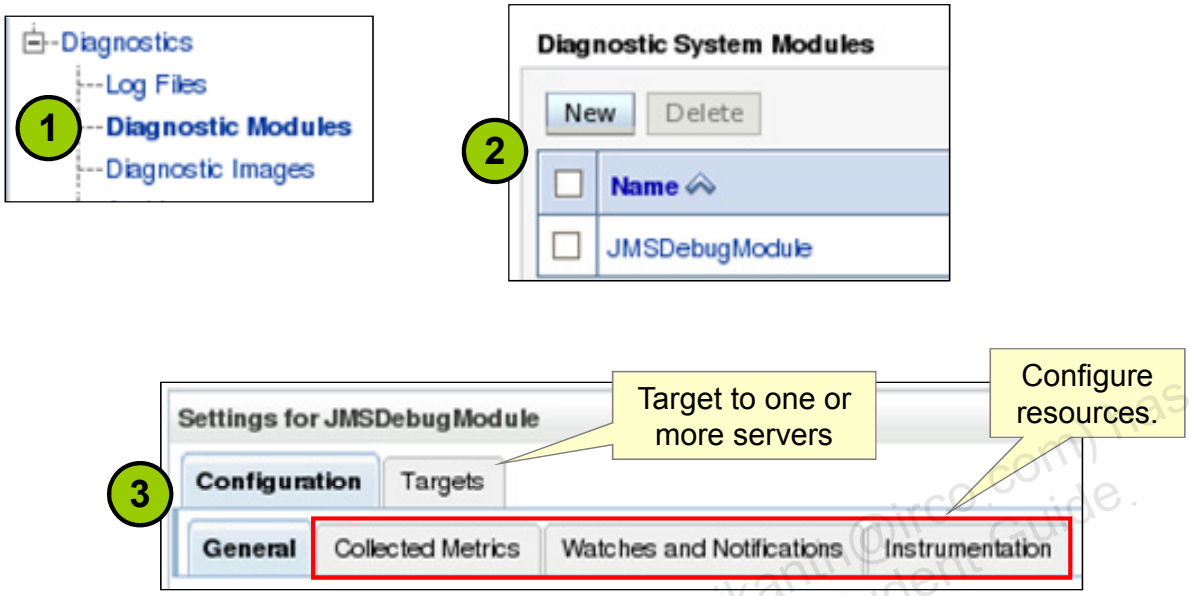


Configuring Server Diagnostic Archives

Log files are archived as human-readable files. Events and harvested data are archived in binary format, in a WebLogic persistent store or in a database.

1. In the left pane, expand Diagnostics and select Archives.
2. Click the name of the server for which you want to configure diagnostic archive settings.
3. Select one of the following archive types from the Type list:
   - Select File Store to persist data to the file system. If you choose this option, enter the directory in the Directory field.
   - Select JDBC to persist data to a database. If you choose this option, select the JDBC data source from the Data Source list. You must first configure JDBC connectivity to use this option.
4. Select or deselect Data Retirement Enabled to enable or disable data retirement for this server instance. In the Preferred Store Size field, enter a maximum data file size, in megabytes. When this size is exceeded, enough of the oldest records in the store are removed to reduce the size of the store below the maximum. In the Store Size Check Period field, enter the interval, in minutes, between the times when the store will be checked to see if it has exceeded the preferred store size.

Creating a Diagnostic Module



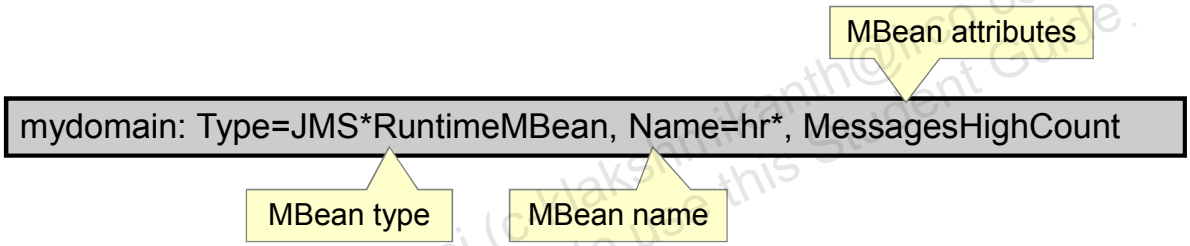
Creating a Diagnostic Module

To configure and use the Instrumentation, Harvester, and Watch and Notification components at the server level, you must first create a system resource called a “diagnostic system module.” System modules are globally available for targeting to servers and clusters configured in a domain. But at most, only one diagnostic system module can be targeted to any given server or cluster.

1. In the left pane, expand Diagnostics and select Diagnostic Modules.
2. Click New. Enter a name for the module and, optionally, enter a description. Then click OK.
3. Use the various Configuration tabs to add diagnostic components to this module.
4. To target the module to a server or cluster, click the Targets tab.

Metric Collectors

- A metric collector definition for the Harvester includes:
  - The runtime MBean type to query
  - The specific MBean instance names to query (all instances, by default)
  - The MBean attributes to collect (all attributes, by default)
  - How often to sample data
- Alternatively, use the MBean expression syntax, which also supports wildcards and complex attributes.



Metric Collectors

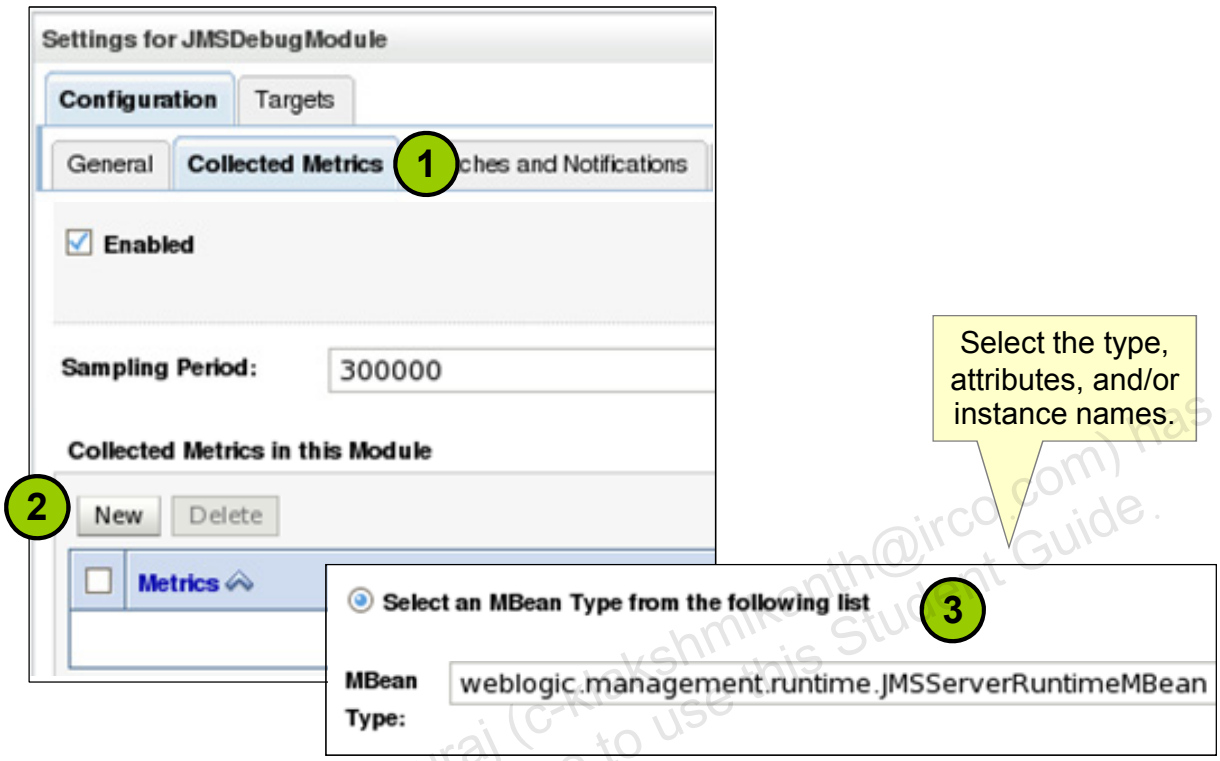
Harvesting metrics is the process of gathering data that is useful for monitoring the system state and performance. Metrics are exposed to WLDF as attributes on qualified MBeans. The Harvester gathers values from selected MBean attributes at a specified sampling rate. Therefore, you can track potentially fluctuating values over time. For custom MBeans, the MBean must be currently registered with the JMX server.

You can configure the Harvester to harvest data from named MBean types, instances, and attributes. If only a type is specified, data is collected from all attributes in all instances of the specified type. If only a type and attributes are specified, data is collected from all instances of the specified type.

The sample period specifies the time between each cycle. For example, if the Harvester begins execution at time T, and the sample period is I, the next harvest cycle begins at T+I. If a cycle takes A seconds to complete and if A exceeds I, then the next cycle begins at T+A. If this occurs, the Harvester tries to start the next cycle sooner, to ensure that the average interval is I.

WLDF allows for the use of wildcards (*) in type names, instance names, and attribute specifications. WLDF also supports nested attributes using a dot delimiter, as well as complex attributes such as arrays and maps. WLDF watch expressions also support similar capabilities.

Configuring a Metric Collector



Configuring a Metric Collector

Metrics are configured and collected in the scope of a diagnostic system module targeted to one or more server instances. Therefore, to collect metrics, you must first create a diagnostic system module.

1. Click the name of the module for which you want to configure metric collection. Then click Configuration > Collected Metrics.
2. To enable or disable all metric collection for this module, select or deselect the Enabled check box. To set the period between samples, enter the period (in milliseconds) in the Sampling Period field. To define a new collected metric, click New.
3. From the MBean Server Location drop-down list, select either DomainRuntime or ServerRuntime. Then click Next. Select an MBean that you want to monitor from the MBean Type list. Then click Next again.
4. In the Collected Attributes section, select one or more attributes from the Available list and move them to the Chosen list (default is all attributes). Click Next.
5. In the Collected Instances section, select one or more instances from the Available list and move them to the Chosen list (default is all instances). Click Finish.
6. Click Save.

Watches and Notifications

- A WLDF *watch*:
  - Inspects data generated from metric collectors, events generated from monitors, or server log files
  - Compares data to one or more conditions or “rules”
  - Triggers one or more *notifications*
- Available notification types include:
  - JMS
  - Email
  - SNMP trap
  - Diagnostic image capture

Watches and Notifications

A watch identifies a situation that you want to trap for monitoring or diagnostic purposes. You can configure watches to analyze log records, data events, and harvested metrics. A watch is specified as a watch rule, which includes a rule expression, an alarm setting, and one or more notification handlers. A notification is an action that is taken when a watch rule expression evaluates to true. You must associate a watch with a notification for a useful diagnostic activity to occur, for example, to notify an administrator about specified states or activities in a running server.

Log and instrumentation watches are triggered in real time, whereas harvester watches are triggered only after the current harvest cycle completes.

Watches and notifications are configured separately from each other. A notification can be associated with multiple watches, and a watch can be associated with multiple notifications. This provides the flexibility to recombine and reuse watches and notifications, according to current needs. Each watch and notification can be individually enabled and disabled as well.

A complete watch and notification configuration includes settings for one or more watches, one or more notifications, and any underlying configurations required for the notification media, for example, the SNMP configuration required for an SNMP-based notification.

Configuring a Watch

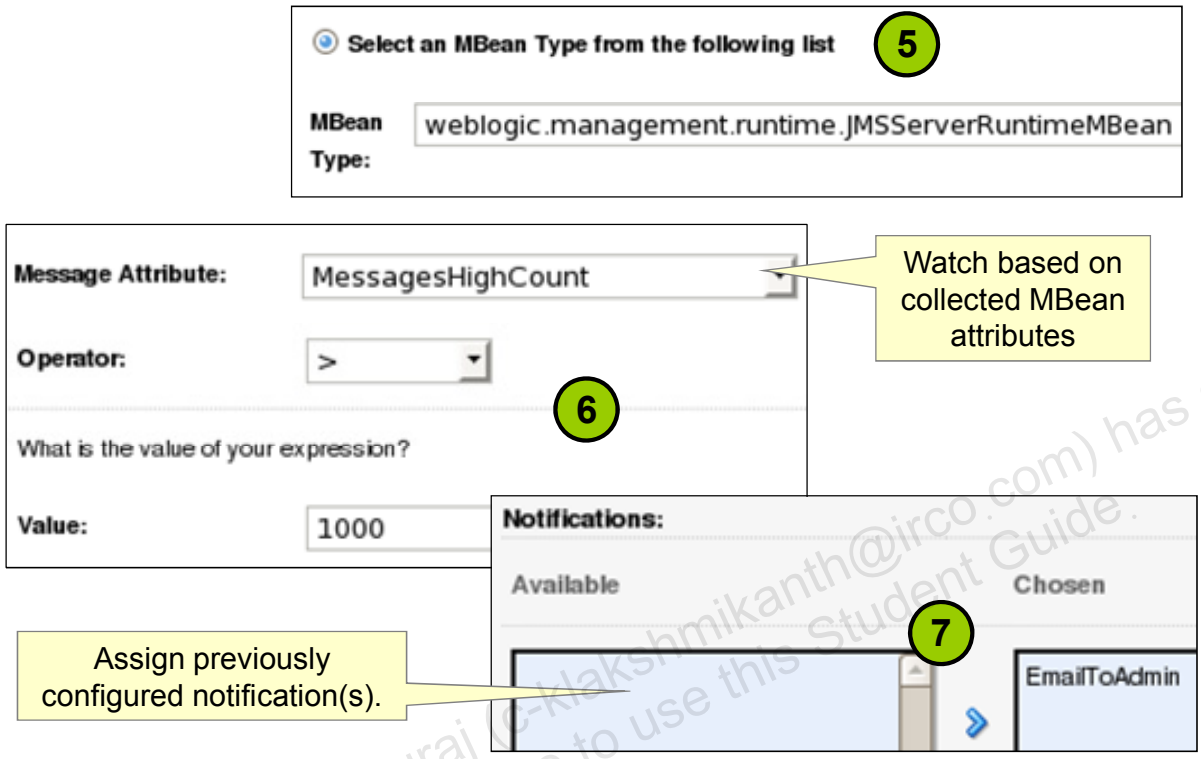
Configuring a Watch

A collected metric watch can monitor any runtime MBean in the local runtime MBean server. Log watches monitor the occurrence of specific messages and/or strings in the server log. Watches contain a severity value that is passed through to the recipients of notifications. Instrumentation watches are triggered as a result of the event being posted that matches some criteria.

The example above depicts a collected metric watch:

1. Click the name of the module for which you want to create a watch. Then click Configuration > Watches and Notifications.
2. In the Watches section, click New.
3. Enter a name for the watch in the Watch Name field. Then select a Watch Type. To enable or disable the watch, select or deselect the Enable Watch check box. Then click Next.
4. Click the Add Expressions button to construct one or more watch expressions. Expressions can be entered manually using the WLDF expression language or constructed graphically. To group two or more expressions, select the check boxes adjacent to the expressions that you want to group and click Combine. To reorder expressions, select the check boxes adjacent to the expressions you want to move and click Move Up or Move Down.

Configuring a Watch



Configuring a Watch (continued)

5. For each watch expression, select either DomainRuntime or ServerRuntime from the MBean Server Location drop-down. Then click Next. Select an instance from the Instance drop-down and click Next again.
6. Select an attribute from the Message Attribute list and an operator from the Operator list, and enter a value with which to compare the attribute using the Value field. The value must be an appropriate value for the attribute chosen above. Click Next.
7. Select and move one or more existing Notifications from the Available list to the Chosen list. Then click Finish.

WLDF WLST Examples

Capture a server diagnostic image:

```
serverRuntime()
wldfCapture = getMBean('WLDFRuntime/WLDFRuntime/
   WLDFImageRuntime/Image')
wldfCapture.captureImage('logs/diagnostic_images',30)
```

Create a diagnostic module and metric collector (harvester):

```
module = cmo.createWLDFSystemResource('JMSDebugModule')
module.addTarget(getMBean('/Servers/serverA'))
harvester = getMBean('/WLDFSystemResources/JMSDebugModule/
   WLDFResource/JMSDebugModule/Harvester/JMSDebugModule')
harvester.setSamplePeriod(300000)
harvester.setEnabled(true)
harvestType = harvester.createHarvestedType
   ('weblogic.management.runtime.JMSServerRuntimeMBean')
harvestType.setEnabled(true)
harvestType.setHarvestedAttributes(['MessagesHighCount'])
```

ORACLE

WLDF WLST Examples

Additional WLDF WLST examples can be found in the WLDF guide in the product documentation.

Refer to the following MBeans:

- `WLDFResourceBean`
- `WLDFHarvesterBean` (a subclass of `WLDFResourceBean`)
- `WLDFHarvestedTypeBean` (a component of `WLDFHarvesterBean`)
- `WLDFInstrumentationBean` (a subclass of `WLDFResourceBean`)
- `WLDFInstrumentationMonitorBean` (a component of `WLDFInstrumentationBean`)
- `WLDFWatchNotificationBean` (a subclass of `WLDFResourceBean`)
- `WLDFWatchBean` (a component of `WLDFWatchNotificationBean`)
- `WLDFNotificationBean` (abstract; subclasses exist for each notification type)
- `WLDFRuntimeMBean`
- `WLDFImageRuntimeMBean` (a component of `WLDFRuntimeMBean`)

Sample WLDF Framework

- WLS ships with a collection of sample WLDF WLST scripts found at `<WL_HOME>/samples/server/examples/src/examples/diagnostics`.
- These scripts provide a framework to:
  - Enable/disable metric collectors and watches for WLS subsystems (JDBC, JMS, Web, EJB, and so on)
  - Enable/disable notifications for sample watches

Sample WLDF Framework

Some users find it difficult to determine what WLS metrics to collect, especially those new to JMX and/or WLDF. WLS includes some WLST scripts and utilities that provide a starting point for creating common WLDF configurations, referred to as “profiles.” Each profile is represented by a Jython class that encapsulates the MBean harvester and watch configurations for a specific WLS subsystem. These class definitions can then be modified or extended to suit your individual needs. In addition, you can use the framework of utility classes separately and outside the context of profile configuration while working with WLDF from the WLST command-line.

`WLDFResource.py` provides a core set of utility classes in Jython for manipulating instances of the WLDFSystemResourceMBean and its constituent bean objects. This file also provides various helper functions that are used by other classes and scripts in this framework.

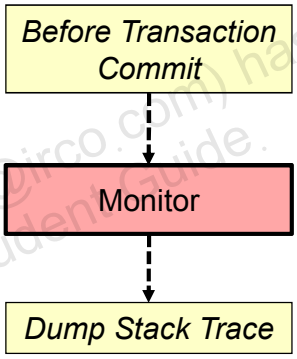
`WLDFProfiles.py` contains the set of Jython classes that represent the predefined profiles, as well as a utility class, `WLDFProfileManager`, for enabling or disabling groups of these profiles. For convenience, various “enable” scripts are included for each profile, such as enableEJBProfile.py and enableJTAProfile.py. Each also has a corresponding “disable” script, which will remove the configured MBean instances associated with the profile.

Sample WLDF Framework (continued)

The scripts accept command-line arguments in the form of name/value pairs to override common settings and parameters, such as the server URL, username, password, WLDF module name, and harvester period.

Instrumentation

- Diagnostic monitors listen for events generated at specific code locations or “pointcuts” associated with:
  - WebLogic Server subsystems (predefined)
  - Application code (custom)
- Events trigger actions such as recording:
  - The raw event data (trace)
  - The current Java stack
  - The current Java threads
  - The current method arguments
  - The time to complete the event
- Actions are recorded in the event data archive.



Instrumentation

A diagnostic monitor is a dynamically manageable unit of diagnostic code that is inserted into server or application code at specific locations. You define monitors by scope (system or application) and type (standard, delegating, or custom). WLDF provides a library of predefined diagnostic monitors and actions. You can also create application-scoped custom monitors, where you control the locations where diagnostic code is inserted in the application.

Instrumentation code is inserted into (or “woven” into) server and application code at precise locations. A joinpoint is a specific location in a class, for example the entry and/or exit point of a method or a call site within a method. A pointcut is an expression that specifies a set of joinpoints, for example all methods related to scheduling, starting, and executing work items. A diagnostic location is the position relative to a joinpoint where the diagnostic activity will take place. Diagnostic locations are “before,” “after,” and “around.”

Diagnostic actions execute diagnostic code that is appropriate for the associated delegating or custom monitor (standard monitors have predefined actions). In order for a delegating or custom monitor to perform any useful work, you must configure at least one action for the monitor. Actions must be correctly matched with monitors. For example, the `TraceElapsedTime action` is compatible with a delegating or custom monitor whose diagnostic location type is “around.”

Instrumentation (continued)

A `TraceAction` generates a trace event at the affected location in the program execution. A trace is similar to a log message and includes standard information such as the timestamp, current server, user ID, and transaction ID. In addition, a trace includes the location in the code from where the action was called.

A `DisplayArgumentsAction` generates an instrumentation event at the affected location in the program execution to capture method arguments or a return value. When attached to "before" monitors, the instrumentation event captures input arguments to the joinpoint (for example, method arguments). When attached to "after" monitors, the instrumentation event captures the return value from the joinpoint.

A `TraceElapsedTimeAction` generates two events: one before and one after the location in the program execution. This action captures the timestamps before and after the execution of an associated joinpoint. It then computes the elapsed time by computing the difference. It generates an instrumentation event that is dispatched to the events archive. The elapsed time is stored as event payload.

A `StackDumpAction` generates an instrumentation event at the affected location in the program execution to capture a stack dump.

A `ThreadDumpAction` generates an instrumentation event at the affected location in the program execution to capture a thread dump, if the underlying VM supports it. JDK 1.5 (Oracle JRockit and Sun) supports this action. It is ignored when used with other JVMs.

A `MethodInvocationStatisticsAction` computes method invocation statistics in memory without persisting an event for each invocation. It makes the collected information available through the `InstrumentationRuntimeMBean`. The collected information is consumable by the Harvester and the Watch-Notifications components. This makes it possible to create watch rules that can combine request information from the instrumentation system and metric information from other runtime MBeans.

The `MethodMemoryAllocationStatisticsAction` uses the JRockit API that tracks the number of bytes allocated by a thread during a method call. Statistics are kept in-memory on the memory allocations, and instrumentation events are not created by this action. This action is very similar to `MethodInvocationStatisticsAction`, except that the statistics tracked by it are related to the memory allocated within a method call.

Working with Diagnostic Monitors

- Predefined system monitors include:
  - JDBC: Get connection, execute statements
  - JMS: Produce/consume messages
  - JTA: Transaction commit/rollback
  - Servlet: Create/update HTTP session
  - EJB: Call method
  - JNDI: Object lookup
- Some predefined monitors are only supported in application-scoped diagnostic modules (`weblogic-diagnostics.xml`)
- Work with developers to define custom monitors (pointcuts and actions) for application code.

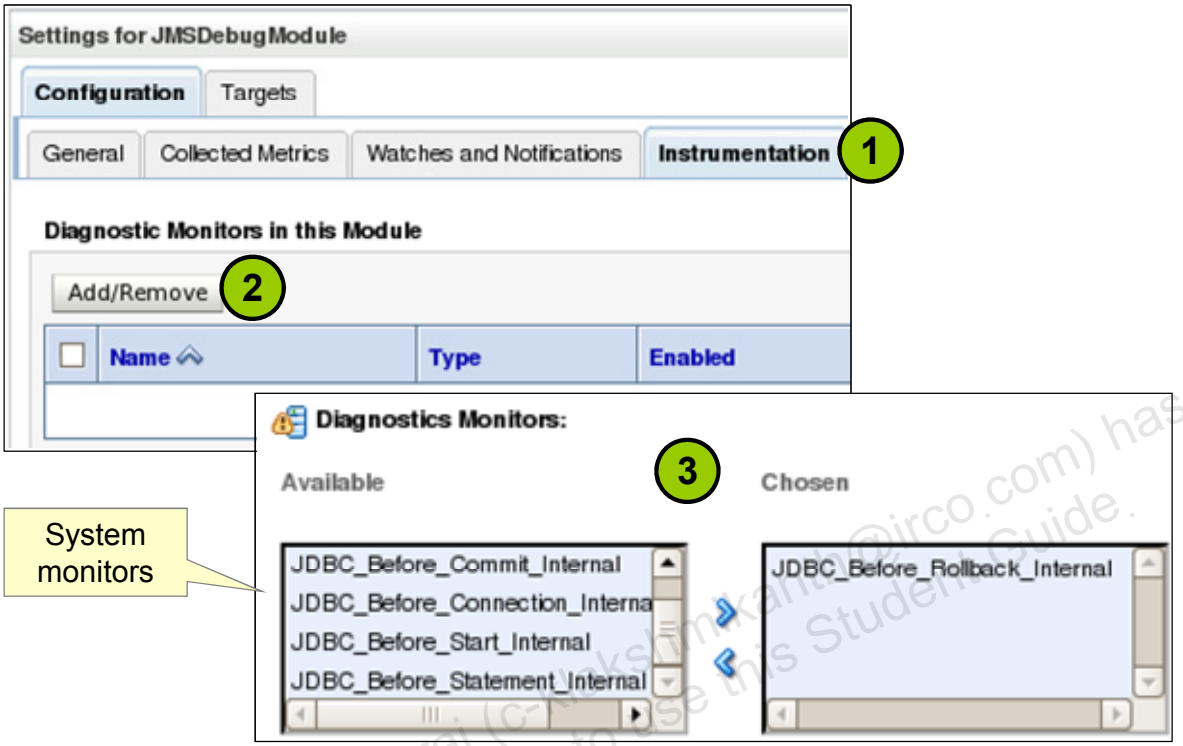
Working with Diagnostic Monitors

Diagnostic monitors are broadly classified as server-scoped and application-scoped monitors. The former can be used to instrument WebLogic Server classes. You use the latter to instrument application classes. Except for the `DyeInjection` monitor, all monitors are delegating monitors, that is, they do not have a built-in diagnostic action. Instead, they delegate to actions attached to them to perform diagnostic activity. Refer to the WLDF Instrumentation Library in the documentation for a list of all available predefined monitors.

All monitors are preconfigured with their respective pointcuts. However, the actual locations affected by them may vary depending on the classes they instrument. For example, the `Servlet_Before_Service` monitor adds diagnostic code at the entry of servlet or Java server page (JSP) service methods at different locations in different servlet implementations.

Application-scoped instrumentation is configured with a `weblogic-diagnostics.xml` descriptor file. Place the file in the application's archive under the `/META-INF` folder. When the archive is deployed, the instrumentation is automatically inserted when the application is loaded. Custom monitors defined in this file include pointcut expressions to control where diagnostics actions are invoked within application code.

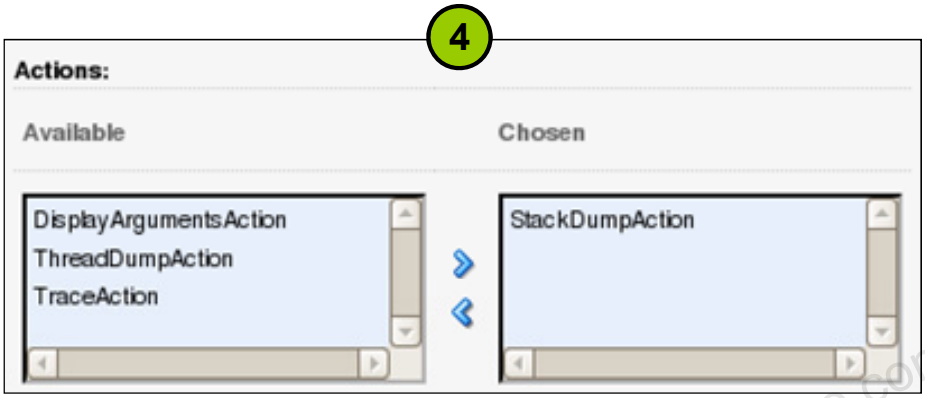
Configuring a System-Scoped Monitor



Configuring a Diagnostic Monitor

1. Click the name of the module to which you want to add diagnostic monitors. Then click Configuration > Instrumentation.
2. Click the Add/Remove button.
3. To enable or disable the monitor, select or deselect Enabled. Then locate the Diagnostic Monitors section and select one or more monitors from the Available list. Click the right arrow button to move the monitors to the Chosen list. The name of the monitor indicates where in the flow of execution the monitoring will take place. If you are editing a monitor other than a DyeInjection monitor and you want to add dye filtering to the monitor, click Enable Dye Filtering to enable it.

Configuring a System-Scoped Monitor



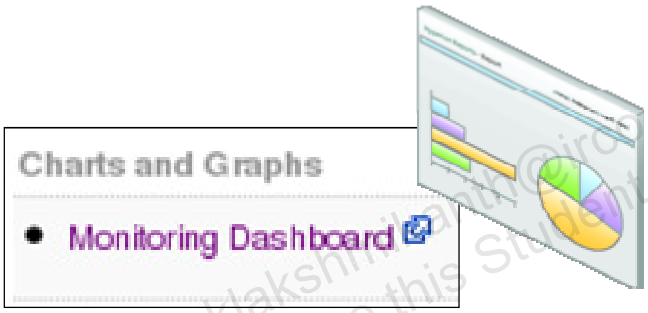
Configuring a Diagnostic Monitor (continued)

4. If you are editing a delegating monitor and you want to add Actions, click the desired actions in the Available list and move them to the Chosen list.
5. Click Finish.

WLDF Monitoring Dashboard

The monitoring dashboard:

- Is part of the administration console
- Monitors selected MBean metrics
- Provides various chart and graph types
- Displays historical trends for metrics that are being collected by a WLDF module



WLDF Console Extension

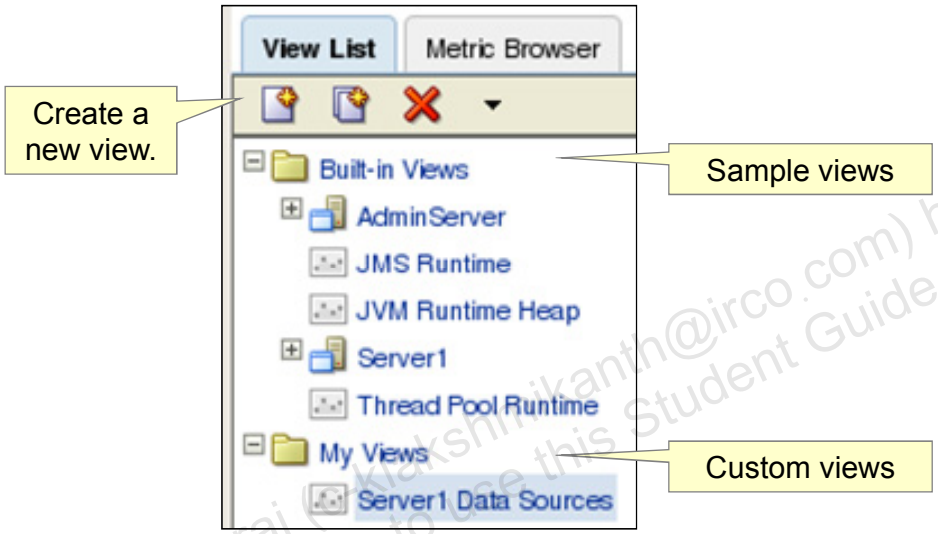
The Monitoring Dashboard provides views and tools for graphically presenting diagnostic data about servers and applications running on them. The underlying functionality for generating, retrieving, and persisting diagnostic data is provided by the WebLogic Diagnostics Framework. The Monitoring Dashboard provides additional tools for presenting that data in charts and graphs.

You can launch the Monitoring Dashboard from the WebLogic Server Administration Console, or you can run it separately in a Web browser. The Monitoring Dashboard is always displayed in its own tab, or window, depending on the preferences you have set for your browser. You do not need to be logged in to the Administration Console to use the Monitoring Dashboard; but if you are not logged in, you are prompted for your username and password credentials.

The diagnostic data displayed by the Monitoring Dashboard consists of runtime MBean attributes with numeric or Boolean values that are useful to measure, either as their current values or as their changes over time. These values, referred to in the Monitoring Dashboard as metrics, originate from one or more runtime MBean instances from one or more servers in the domain. The Monitoring Dashboard obtains metrics from two sources. First, it gathers metric data directly from the active MBean instances. Second, it gathers data from any diagnostic archives that have been automatically collected by the WLDF Harvester component.

Views, Charts, and Graphs

- A diagnostic *view* is a collection of charts.
- Each *chart* contains labels and controls for displaying one or more graphs.



ORACLE

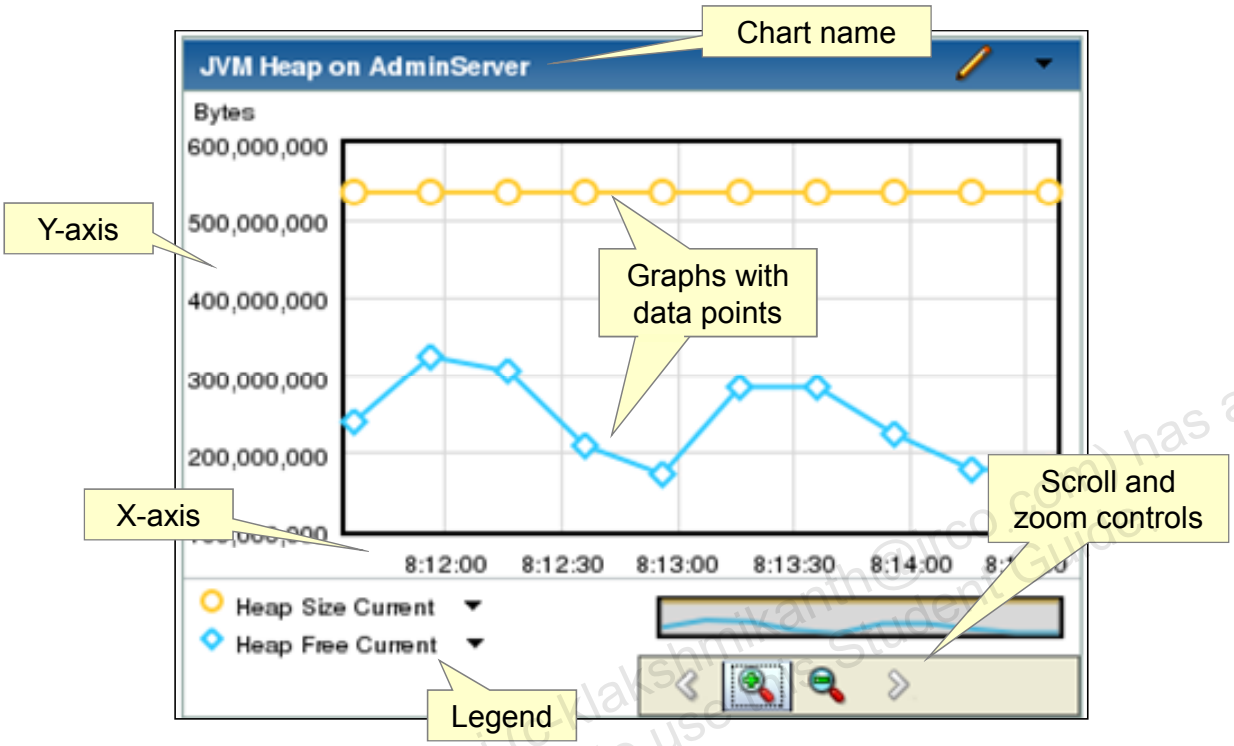
Views, Charts, and Graphs

The Monitoring Dashboard has two main panels: the explorer panel and the view display panel. The explorer panel consists of two tabs. The View List tab displays the set of existing built-in and custom views. It also contains controls for creating, copying, renaming, and deleting views. The Metric Browser tab provides a means to navigate to and select the specific MBean instance attributes whose metric values you want to display in a chart in a view.

A view is a collection of one or more charts, which display captured monitoring and diagnostic data. Each chart contains a legend, labels, and controls for identifying and displaying the data. A chart can display data from one or more data sources from one or more servers in the domain. Each data source is displayed as a distinct graph, which shows the values of data points over a time span. A chart can include graphs based on data from one or more servers in the domain.

Built-in views are dynamic. For example, if four servers are running, the set of available built-in views and its charts are related to those four servers. If five servers are running, then the set of built-in views and its charts expands for each additional server. In addition, if the number of running server instances changes while you are using dashboard (for example, a server is started or stopped), and you want to see the new built-in views for the current set of running server instances, refresh the view list by selecting Refresh from the View List menu.

Anatomy of a Chart



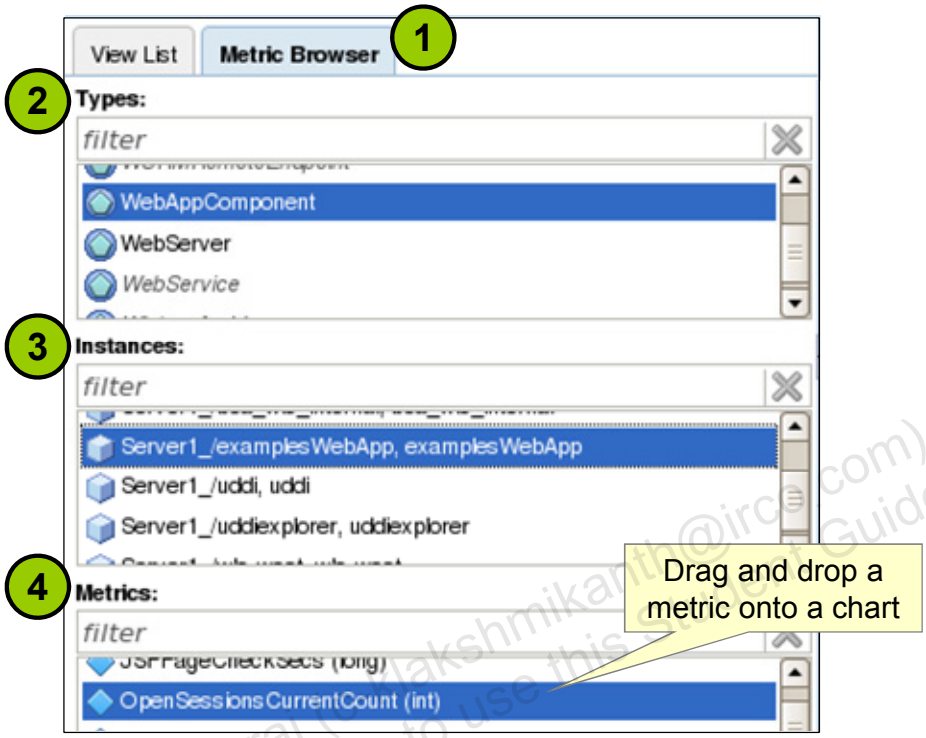
Anatomy of a Chart

A chart is one or more graphs that show data points over a specified time span. The chart depicted in the slide consists of two graphs: HeapSizeCurrent and HeapFreeCurrent.

When working with a view, you can do the following:

- Start and stop data collection for charts in a view.
- Add charts to views.
- Add graphs to charts.
- Merge charts.
- Move a graph to a different chart.
- Pan along the time axis.
- Zoom in and out of a chart (change the time axis).

Creating a New Graph



Creating a New Graph

1. Click the Metric Browser tab. Select one of the available servers in this domain and click Go.
2. Select a specific MBean Type to constrain the list of MBean instances. When you enter a filter string into any of the list boxes, you constrain the list contents to include only the items that match the filter.
3. Select a specific MBean Instance to constrain the list of available metrics to display.
4. Drag and drop an option from the Metrics list onto a chart in the right view panel to display it as a new graph in the chart.

As a convenience for selecting metrics that have been collected by the Harvester, the Metric Browser includes the Collected Metrics Only button. To see metrics for all runtime MBean types regardless of whether instances of them are currently active, select Include All Types. To determine whether a metric was collected by the harvester, select the metric, or leave the mouse positioned over it. A note window is displayed that provides information about the metric, including whether or not it is a collected metric.

Section Summary

In this section, you should have learned how to:

- Describe some capabilities of WLDF
- Capture a diagnostic image
- Collect and record MBean metrics
- Use monitors and actions to collect diagnostic information at specific server events
- List some features of the WLDF monitoring dashboard

Practice 18-1
Configure and Monitor Diagnostic Data

This practice covers the following topics:

- Installing the WLDF console extension
- Capturing a server diagnostic image
- Collecting and recording MBean metrics by using WLDF
- Generating diagnostic events by using WLDF monitors
- Utilizing standard diagnostic views in the WLDF console
- Creating a custom view in the WLDF console

Road Map

- Monitoring Concepts
- WLDF Fundamentals
- WLS SNMP
  - OIDs and MIBs
  - Message Types
  - Agents
  - SNMP Channels
  - Custom Notifications
  - Command-Line Utility
- WLS Debugging

Simple Network Management Protocol (SNMP)

- SNMP:
  - Is a simple protocol for managing and monitoring distributed, heterogeneous devices or “agents”
  - Allows enterprises to have a centralized management infrastructure
  - Is well supported throughout the industry
  - Can be run over UDP (default) or TCP
- WLS provides an SNMP agent:
  - For monitoring a server’s configuration and runtime state
  - That supports SNMP v1, v2, and v3

Simple Network Management Protocol (SNMP)

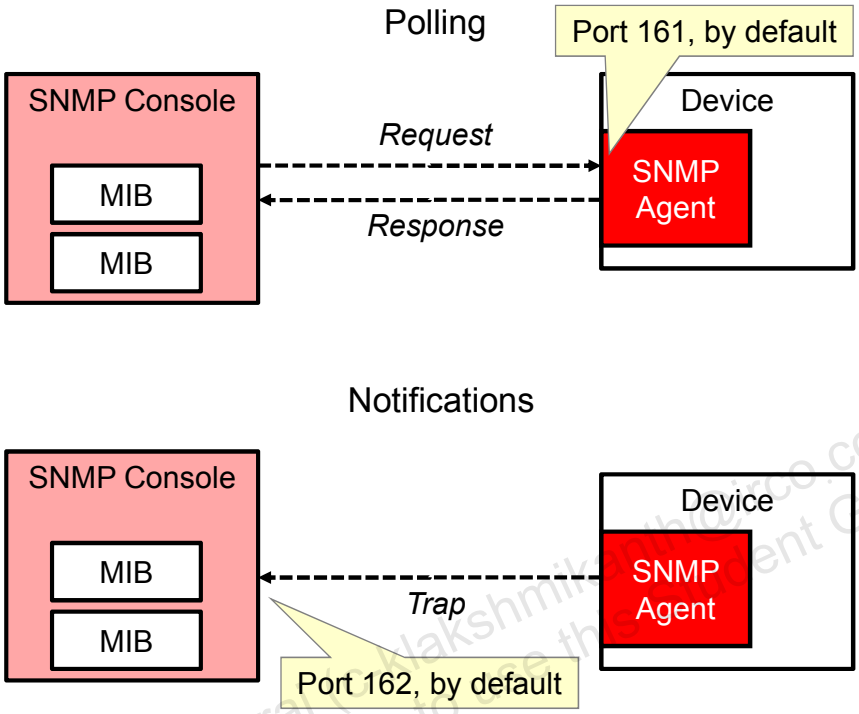
SNMP is used in network management systems to monitor network-attached devices for conditions that warrant administrative attention. SNMP exposes management data in the form of variables on the managed systems, which describe the system configuration. These variables can then be queried (and sometimes set) by managing applications.

In typical SNMP use, there are a number of systems to be managed, and one or more systems managing them. A software component called an agent (see below) runs on each managed system and reports information via SNMP to the managing systems.

With SNMP, a manager sends a request for information about managed resources to an agent. The agent gathers the requested data and returns a response. You can also configure agents to issue unsolicited reports (notifications) to managers when they detect predefined thresholds or conditions on a managed resource.

In practice, SNMP implementations often support multiple versions. SNMP version 1 (SNMPv1) is the initial implementation of the SNMP protocol. SNMPv1 operates over protocols such as User Datagram Protocol (UDP). SNMPv2 revises version 1 and includes improvements in the areas of performance, security, confidentiality, and manager-to-manager communications. SNMPv3 provides three important enhancements: authentication, privacy, and access control.

SNMP Architecture



SNMP Architecture

A managed device is a network node that contains an SNMP agent and that resides on a managed network. Managed devices collect and store management information and make this information available to management systems or consoles using SNMP. Managed devices, sometimes called network elements, can be any type of device including, but not limited to, routers, access servers, switches, bridges, hubs, IP telephones, computer hosts, and printers.

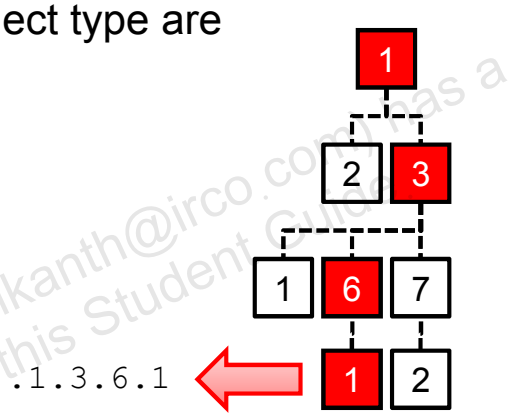
An agent is a network-management software module that resides in a managed device. An agent has local knowledge of management information and translates that information into a form compatible with SNMP.

Typically, SNMP uses UDP ports 161 for the agent and 162 for the manager. The manager may send synchronous polling requests to port 161 to retrieve values for specific variables. However, agents may also publish asynchronous notifications or “traps” to port 162 on a periodic basis or when variables meet certain conditions.

Although SNMP supports requests that update variables with new values, WebLogic Server’s agent gives SNMP managers only read access to its management system.

Object Identifier (OID)

- Similar to JMX MBeans, an SNMP device's manageable objects are organized using a hierarchy.
- Each object is identified using a unique OID, which is:
  - Represented as a series of dot-separated integers
  - Supplied as part of an SNMP poll or trap message
- Multiple instances of the same object type are represented as tables.



Object Identifier (OID)

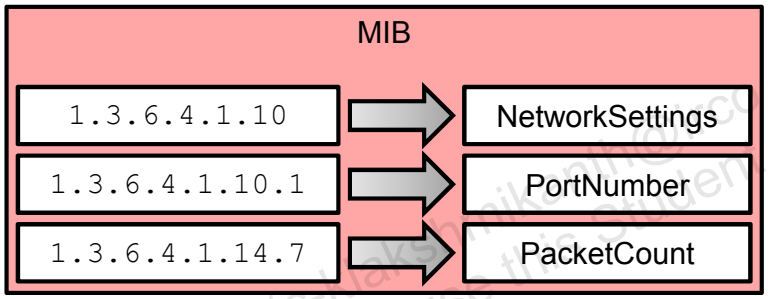
WebLogic Server SNMP agents query the WebLogic Server management system and communicate the results to managers over the SNMP protocol. The WebLogic Server management system exposes management data through a collection of managed beans (MBeans). When a WebLogic Server SNMP agent receives a request from a manager, it determines which MBean corresponds to the SNMP object identifier (OID) in the manager's request. Then it retrieves the data and wraps it in an SNMP response. Similarly, trap messages generated in response to some criteria include an OID that maps to the corresponding MBean being evaluated.

Each OID consists of a left-to-right sequence of integers. This sequence defines the location of the object in the hierarchy and specifies a unique path through the tree to the object. Each node in the path has both a number and a name associated with it. The path .1.3.6.1.4.1 defines the "private.enterprises" OID and each number beneath that node on the tree represents the branches in the tree reserved for a particular vendor, for example, Oracle.

Management Information Base (MIB)

MIB modules:

- Are files that use the ASN.1 format
- Are registered with an SNMP console
- Identify a set of available SNMP objects for a device
- Map numeric OIDs to descriptive names



Management Information Base (MIB)

In telecommunications and computer networking, Abstract Syntax Notation One (ASN.1) is a standard and flexible notation that describes data structures for representing, encoding, transmitting, and decoding data. It provides a set of formal rules for describing the structure of objects that are independent of machine-specific encoding techniques and is a precise, formal notation that removes ambiguities. An adapted subset of ASN.1, Structure of Management Information (SMI), is specified in SNMP to define sets of related objects; these sets are termed Management Information Base (MIB) modules.

Two types of managed objects exist. Scalar objects define a single object instance. Tabular objects define multiple related object instances that are grouped in MIB tables. Tables are composed of zero or more rows, which are indexed in a way that allows SNMP to retrieve or alter an entire row with a single command.

WLS MIB and OIDs

- The WLS SNMP agent supports most standard MBeans and their attributes, including:
  - Domains, servers, JVM, and work managers
  - Applications
  - JDBC and JMS services
- WLS includes two versions of its MIB module:
  - An ASN.1 file found at `<WL_HOME>/server/lib`
  - An XML file found in `weblogic.jar`
- All WLS OIDs begin with `.1.3.6.1.4.1.140.625`.
- The documentation includes an interactive application for browsing and searching the MIB hierarchy.

WLS MIB and OIDs

WebLogic Server exposes a large number of data points in its management system. To organize this data, it provides a hierarchical data model that reflects the collection of services and resources that are available in a domain. The WebLogic Server MIB module reflects this hierarchy. For example, a WebLogic Server domain describes its overall configuration in a tabular managed object called domainTable. This tabular object refers to (contains) a collection of scalar objects, each of which describes some attribute of the domain. For example, domainTable contains a domainServers scalar object that names all servers in the domain. The serverTable object contains a serverDeployments scalar object, which describes all applications currently deployed on a server.

Tabular objects never directly contain object instances (MIB variables). Instead, tabular objects contain scalar objects, and the scalar objects contain variables. For example, if you created two managed servers in a domain named “MS1” and “MS2,” the MIB contains one serverTable object, which in turn contains a serverName object. The serverName object contains two variables that contain the value “MS1” and “MS2.”

Common SNMP Message Types

Type	Description	Versions Supported
GET	Get one or more objects	All
GETNEXT	Get the next sibling object after the given one	All
GETBULK	Gets all objects found in a given ID range	V2,V3
WALK	Get an object and all of its children	V2,V3
TRAP	Notification object sent from device	All
INFORM	TRAP message with acknowledgment	V2,V3

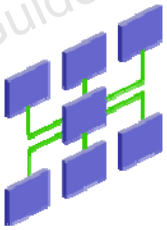
Common SNMP Message Types

SNMPv2 introduced GETBULK in order to retrieve large amounts of management data in a single request. In SNMPv1, this could only be accomplished using a series of GETNEXT requests.

An SNMP agent that uses the SNMPv2 or SNMPv3 protocol can send one of two types of notifications when a monitored attribute crosses a defined threshold. The agent sends a TRAP notification once and assumes that the SNMP manager received the message. The agent sends an INFORM notification and waits for a response from the SNMP manager that indicates the manager has received the message. If the manager does not respond, the agent resends the notification. By default, a WebLogic Server SNMP agent sends TRAP notifications.

WLS SNMP Architecture

- Internally, WLS agents use JMX to monitor servers.
- WLS supports two SNMP models:
  - Centralized
  - Decentralized
- In a centralized architecture, all SNMP communications:
  - Use a single, domain-wide agent
  - Are directed through the administration server
- A new domain is automatically configured with a domain-wide SNMP V1 agent, but it is disabled by default.



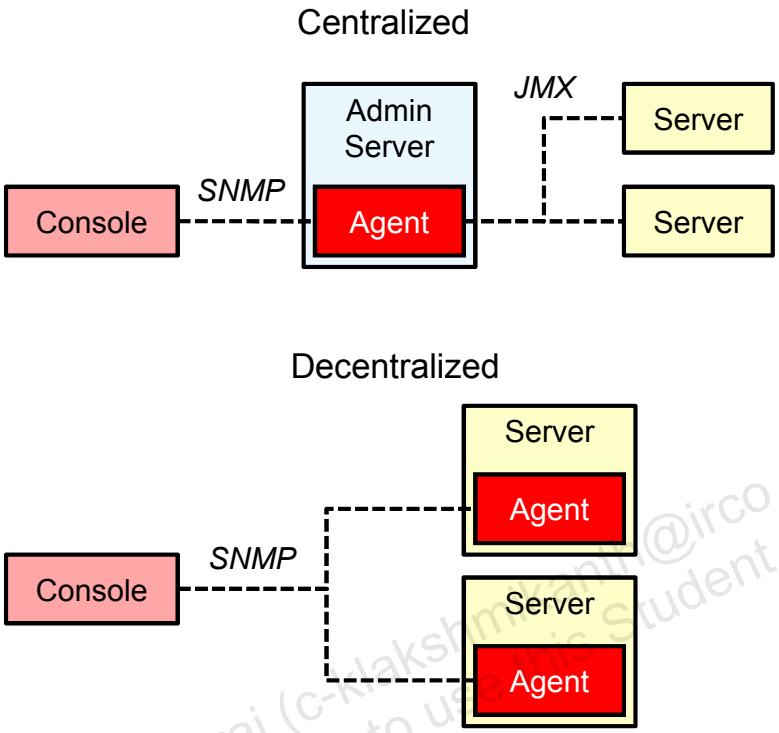
WLS SNMP Architecture

In each WebLogic Server domain, you can create multiple SNMP agents and organize them into a decentralized or centralized model for SNMP monitoring and communication. In a decentralized model, you create SNMP agents on each managed server. SNMP managers communicate with the agents on individual managed servers.

In a centralized model, you create an SNMP agent only on the Administration Server. SNMP managers communicate only with the SNMP agent on the Administration Server and the agent gathers monitoring data from all managed servers in the domain. This model is convenient and enables a single request to retrieve data for the entire domain, but there are some potential drawbacks. If the Administration Server is unavailable, you cannot monitor the domain through SNMP. The model also introduces performance overhead. Because the Domain Runtime MBean server communicates with all managed servers in the domain, it is subject to network latency and increases the amount of memory that the Administration Server uses.

The domain-scoped agent is automatically overridden if you target a server SNMP agent to the Administration Server.

WLS SNMP Architecture



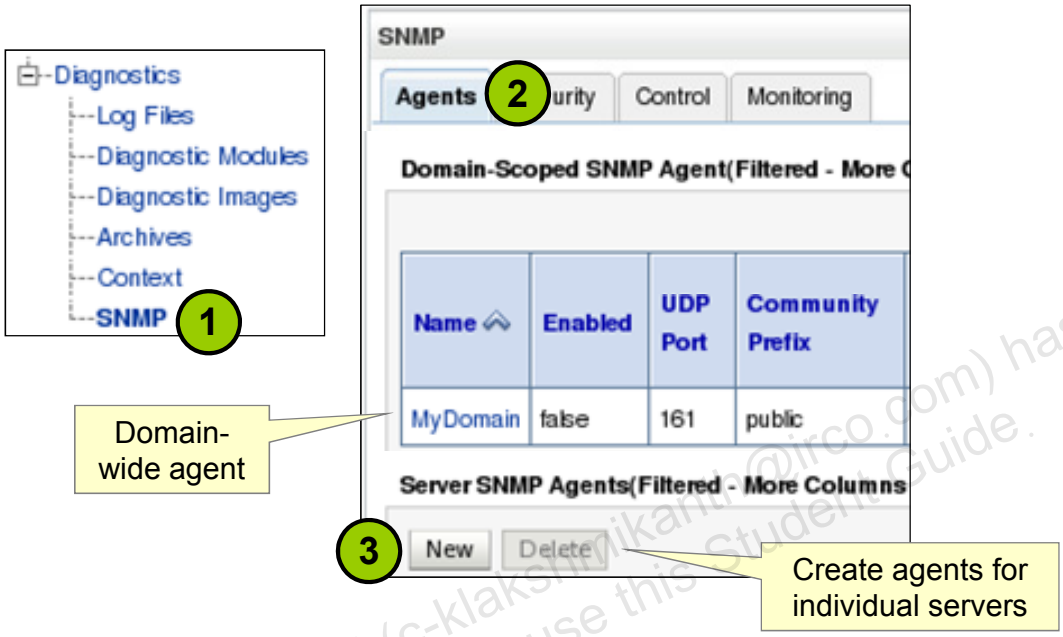
WLS SNMP Architecture (continued)

A WebLogic Server SNMP agent can always communicate with managers using the SNMPv3 protocol. You can configure whether the agent also supports the SNMPv1 and SNMPv2 protocols. While you cannot prevent an agent from receiving SNMPv3 requests, an agent processes only requests from known users that you configure through the WebLogic Server security realm.

If you plan to target the SNMP agent that contains a trap monitor to the Administration Server, specify which servers in the domain you want the monitor to observe. If you target SNMP agents to individual managed servers, you do not need to specify which servers you want to monitor. An SNMP agent on a managed server monitors only its host managed server.

Although not shown here, WebLogic SNMP agents can also function as master agents that forward (proxy) requests to other SNMP agents. To use the master agent functionality of a WebLogic Server agent, you assign branches of the registration tree (OID tree) as the responsibility of other SNMP agents. When an SNMP manager sends a request to a WebLogic SNMP agent, if the OID of the requested object is under the branch of the OID tree assigned to a proxied agent, the WebLogic SNMP agent forwards the request to the proxied agent.

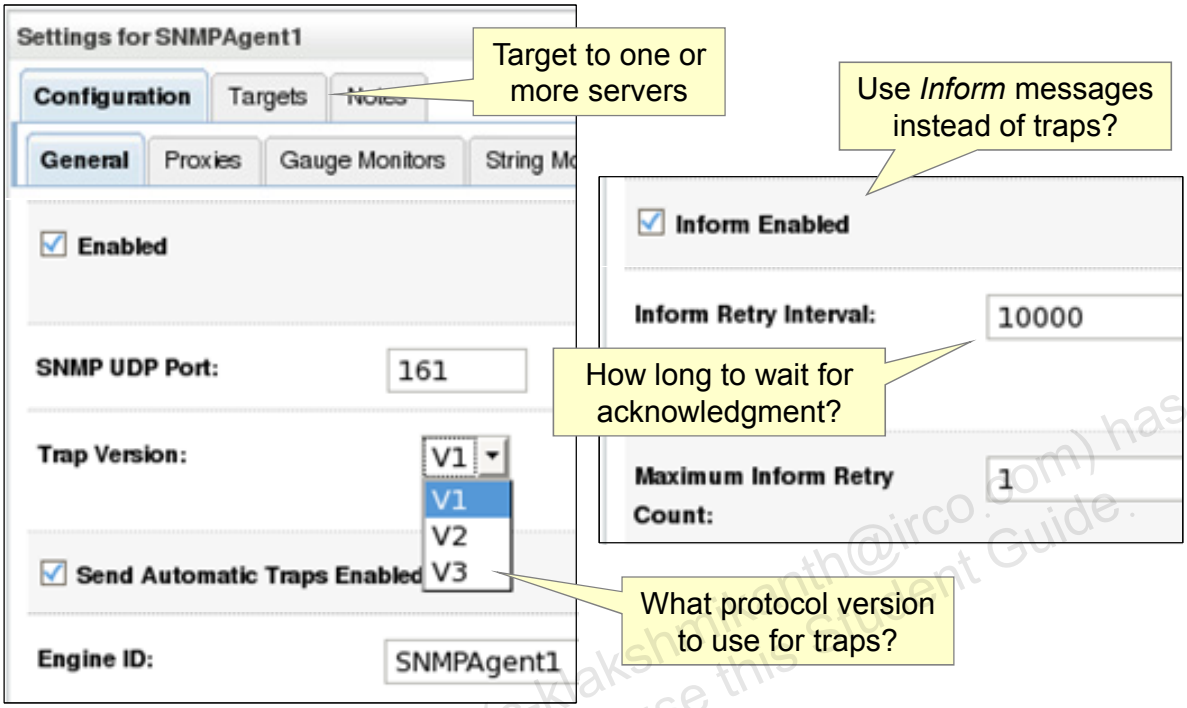
Creating an SNMP Agent



Creating an SNMP Agent

1. Under Domain Structure, expand Diagnostics and select SNMP.
2. Click the Agents tab if not already selected.
3. In the Server SNMP Agents table, click the New button. The SNMP page also displays a Domain SNMP Agent table, which provides access to the default, domain-scoped SNMP agent.
4. Enter a name for the SNMP agent and click the Finish button.
5. Select your new agent and configure it as desired.

Configuring an SNMP Agent



Configuring an SNMP Agent

- **Enabled:** Specifies whether this SNMP agent is enabled.
- **SNMP UDP Port:** The port on which you want this SNMP agent to listen for incoming requests from SNMP managers that use the UDP protocol. If you target this SNMP agent to multiple server instances, and if two or more servers are running on the same computer, WebLogic Server will automatically increment this UDP port value by 1 for each agent. WebLogic Server never assigns port 162, because it is the default port that an agent uses to send notifications. In addition, if any port is already in use, WebLogic Server skips the port and assigns the next available port.
- **Trap Version:** The SNMP notification version that this SNMP agent generates.
- **Send Automatic Traps Enabled:** Specifies whether this SNMP agent sends automatically generated notifications to SNMP managers when servers are started or stopped.
- **Engine ID:** An identifier for this SNMP agent that is unique among all other SNMP agents in the current WebLogic Server domain. If you use SNMPv3 to send messages to this SNMP agent, you must specify the SNMP engine ID when you configure the SNMP manager.

Configuring an SNMP Agent (continued)

- **Inform Enabled:** Configures this SNMP agent to send notifications as an INFORM instead of a TRAP. Requires you to specify the agent's SNMPTrapVersion as SNMPv2 or SNMPv3.
- **Inform Retry Interval:** The number of milliseconds that this SNMP agent will wait for a response to an INFORM notification before resending.

SNMP Channels

To support SNMP over the TCP transport, create a separate network channel.

SNMP Channels

An SNMP agent communicates through a port that accepts UDP traffic and another port that accepts TCP traffic. By default, all TCP traffic uses the host server's listen port. For example, if you target this agent to a server named ManagedServer1, and ManagedServer1 listens for requests on port 7001, then the SNMP agent listens for TCP requests on port 7001. When communicating through a TCP port, WebLogic Server protects SNMP communication from denial of service (DOS) attacks. If you want to separate SNMP TCP traffic from business traffic, you can create a custom network channel.

Create the SNMP channel as you would create any network channel for a server (under the Protocols tab). Any SNMP agent that you target to this WebLogic Server instance will then automatically use the SNMP network channel that you created.

WLS SNMP Notifications

- By default, agents automatically send traps for server startup and shutdown.
- Agents also support custom traps that monitor log messages or MBean attributes.

Monitor Type	Description
Change	Sends a notification every time a *configuration* attribute has a new value
String	Sends a notification if a *runtime* attribute is equal to or different from a specified value
Gauge	Sends a notification if a *runtime* attribute is above a high threshold or below a low threshold
Counter	Sends a notification if a *runtime* attribute exceeds some threshold

WLS SNMP Notifications

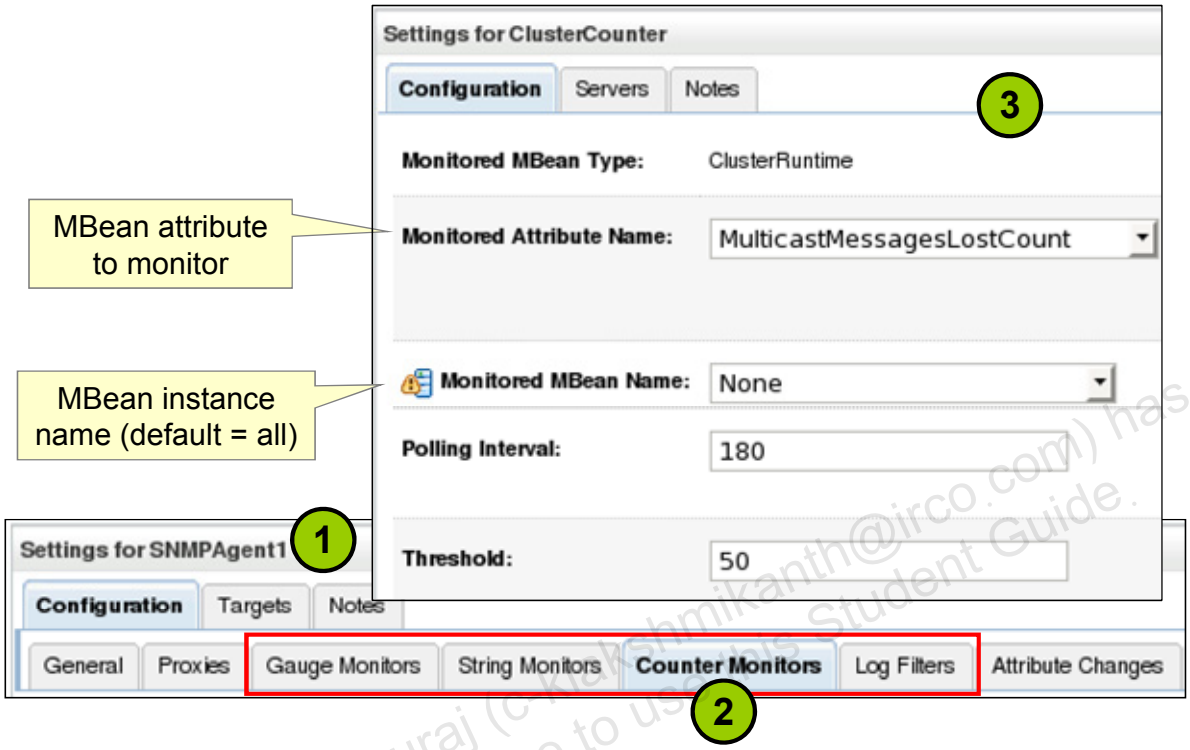
You can configure a WebLogic Server SNMP agent to detect certain thresholds or conditions within a managed resource and send a report (notification) to one or more SNMP managers. WebLogic Server SNMP agents will automatically generate the following notifications without any additional configuration:

- **coldStart:** The WebLogic Server instance that hosts the SNMP agent starts.
- **serverStart:** A WebLogic Server instance that was down is now up. Contains two name-value pairs to identify server start time and the server name.
- **serverShutDown:** A server that was up is now down. Contains two name-value pairs to identify server down time and the server name.

Each server instance saves messages in a local log file and then broadcasts them as JMX notifications. You can set up a WebLogic Server SNMP agent to listen for all of these JMX notifications or you can set up a filter based on criteria such as the severity level, subsystem, or the message text. Log message notifications include these variable bindings: trapTime, trapServerName, trapLogSubsystem, trapLogSeverity, and trapLogMessage.

JMX monitors poll WebLogic Server MBeans at a specified interval and send notifications to an WebLogic SNMP agent when an event that you specify occurs, such as the crossing of a threshold. The SNMP agent generates a notification and sends it to the SNMP managers. Gauge and counter monitors are only applicable to numeric MBean attributes.

Creating Trap Monitors

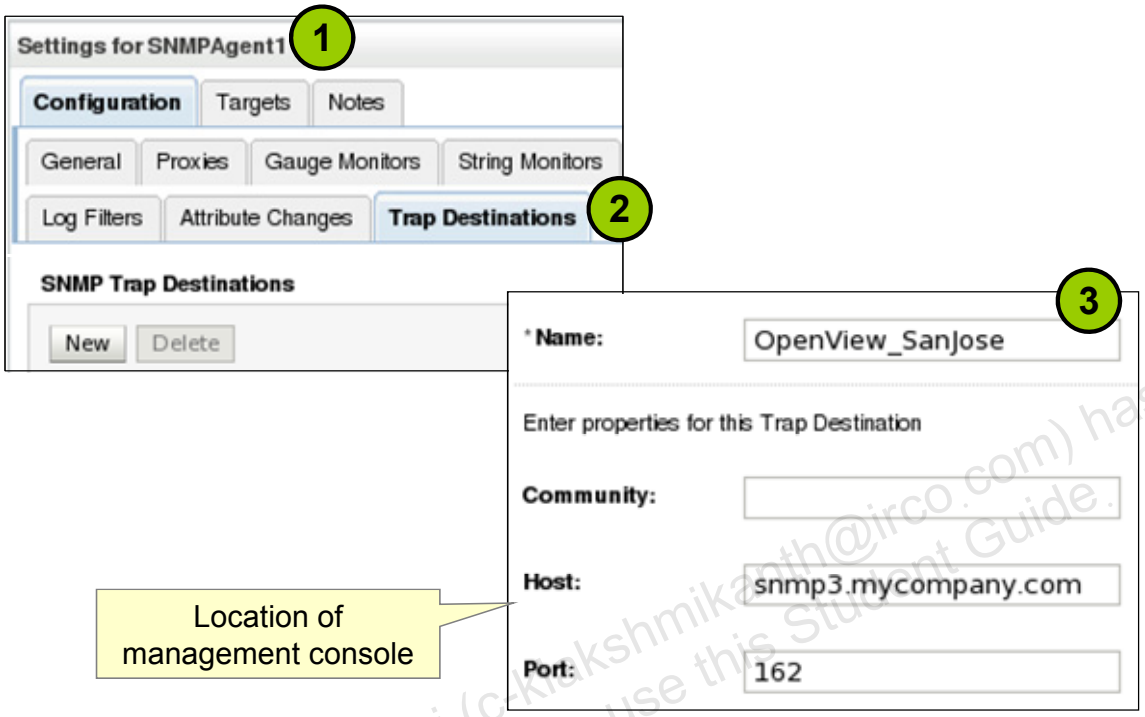


Creating Trap Monitors

A counter monitor periodically polls an attribute whose value is an integer and compares the attribute value with a high threshold that you define. It generates a notification if the attribute value equals or exceeds the high threshold. You can also configure an offset or modulus, which increases or decreases the threshold each time the threshold is crossed.

1. Click the name of an existing SNMP agent.
2. Click Configuration > Counter Monitors. Then click New.
3. Enter the following information:
   - **Name:** The name of the monitor
   - **Monitored MBean Type:** The runtime MBean type that defines the attribute you want to monitor
   - **Monitored MBean Name:** The name of the MBean instance that you want to monitor. If you leave the name undefined, WebLogic Server monitors all instances of the MBean type that you specify in Monitored MBean Type. Alternatively, use the User Entered MBean name field to enter a name directly.
   - **Monitored Attribute Name:** The name of an MBean attribute to monitor. This attribute must be in the WebLogic Server MIB.
   - **Polling Interval:** The frequency (in seconds) that the server checks the attribute value

Creating Trap Destinations



Creating Trap Destinations

Trap destinations specify the SNMP management station to which a WebLogic Server SNMP agent sends notifications, both automatically generated ones and those generated by custom monitors (counters, gauges, and so on).

1. Click the name of an existing SNMP agent.
2. Click Configuration > Trap Destinations. Then click New.
3. Enter the following information:
   - **Name:** The name of this trap destination. This value is for your identification purposes only.
   - **Community:** The “password” that a WebLogic Server SNMP agent sends to the SNMP manager when the agent generates SNMPv1 or SNMPv2 notifications. The community name that you enter in this trap destination must match the name that the SNMP manager defines. For SNMPv3, use the Security Name and Security Level fields instead.
   - **Host:** The DNS name or IP address of the computer on which the SNMP manager is running
   - **Port:** The UDP port on which the SNMP manager is listening

SNMP Security

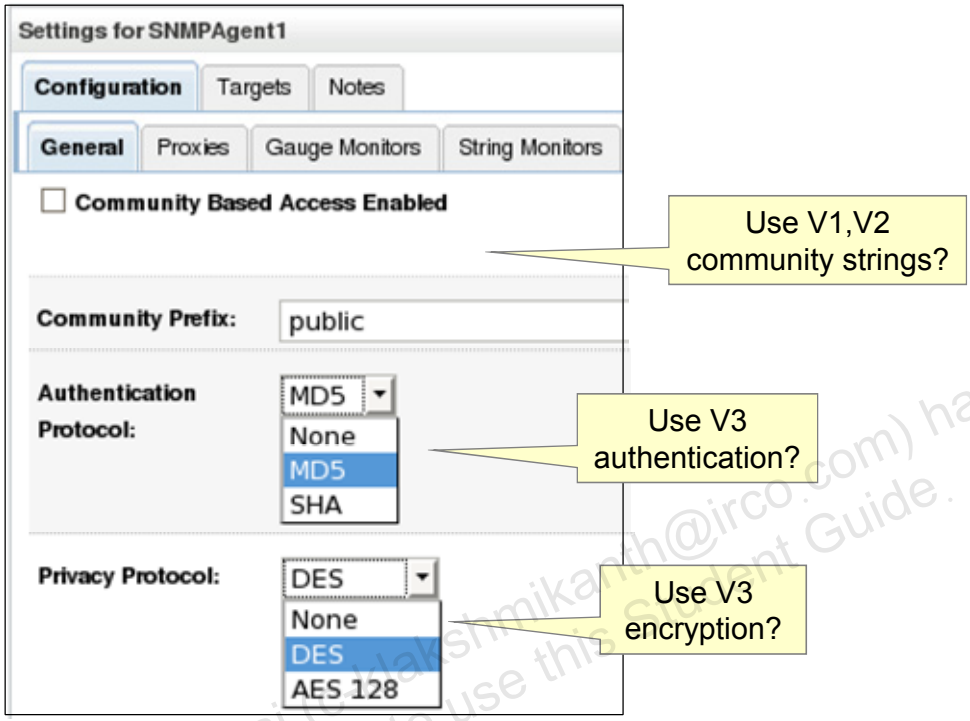
- SNMP V1 and V2 simply support clear text passwords called “communities.”
- SNMPv3 supports:
  - Authentication based on a username/password
  - MD5 or SHA hashing of authentication data to prevent message tampering
  - DES or AES encryption based on a privacy password

SNMP Security

The security features that are available for SNMP depend on which SNMP protocol an agent uses to communicate with managers. To ensure that an SNMP manager requesting data from the WebLogic SNMP agent has permission to obtain the data, and to verify that the agent has permission to send notifications to a target manager, SNMPv1 and SNMPv2 use clear-text passwords called community names.

In the SNMPv3 protocol, both SNMP agent and manager must encode identical credentials in their PDUs for the communication to succeed. The credentials include several tokens: a username, an SNMP engine ID, an authorization protocol, and an optional privacy password, all of which are encrypted before being transported over the network.

Configuring Agent Security



Configuring Agent Security

Because SNMPv1 and SNMPv2 use clear-text passwords, the level of security is weak. If you can use SNMPv3 to communicate with managers, consider disabling SNMPv1 and SNMPv2 by disabling Community Based Access Enabled for each SNMP agent.

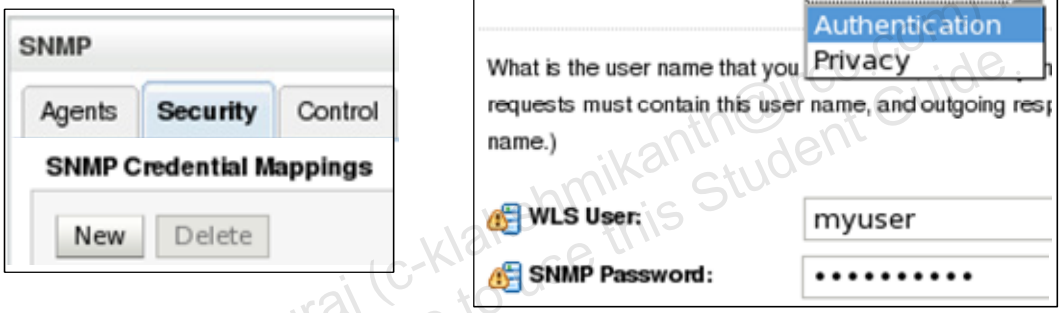
Community Prefix: The password that you want this SNMP agent to use to secure SNMPv1 or v2 communication with SNMP managers. SNMPv3 does not use community names, so this field is ignored.

Authentication Protocol: The protocol that this SNMP agent uses to ensure that only authorized users can request or receive information about your WebLogic Server domain. Applicable only with SNMPv3. The protocol also ensures message integrity and prevents masquerading and reordered, delayed, or replayed messages. If you do not choose an authentication protocol, the SNMP agent does not authenticate incoming SNMPv3 requests; anyone can use SNMPv3 to retrieve information about your WebLogic Server domain.

Privacy Protocol: The protocol that this SNMP agent uses to encrypt and unencrypt messages. If you do not choose a privacy protocol, communication between this agent and managers can be viewed (but not altered) by unauthorized users.

Configuring SNMPv3 Credentials

- Register username/passwords that will be:
  - Granted access to the SNMP agent
  - Included when sending notifications
  - Used as privacy passwords for encryption
- Usernames must map to existing WLS users that have the appropriate monitoring privileges.



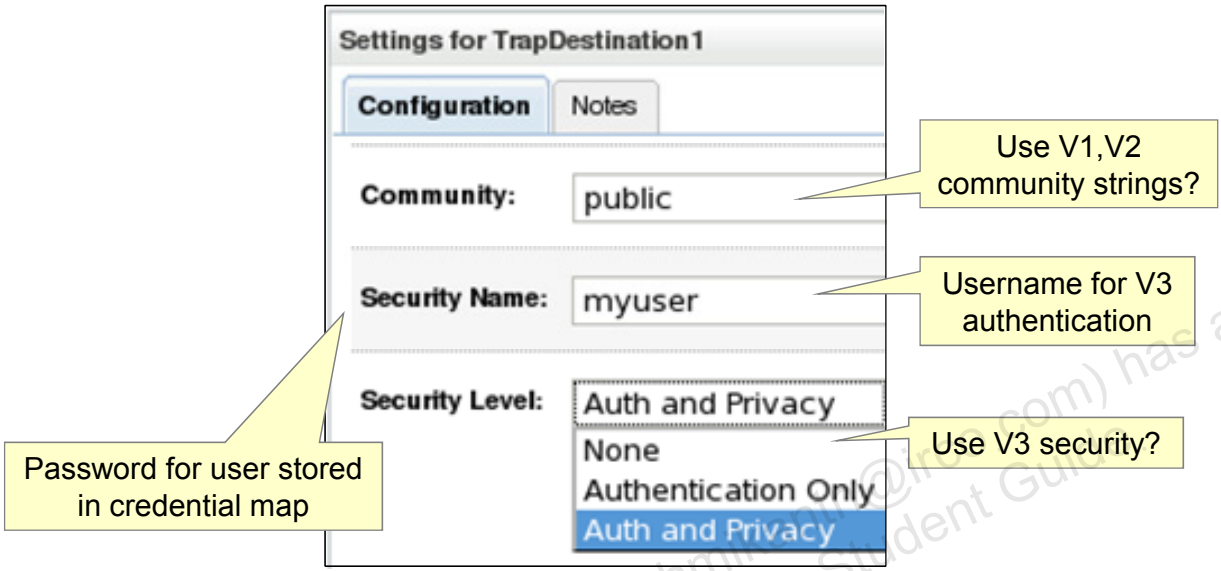
Configuring SNMPv3 Credentials

In WebLogic Server, SNMPv3 agents work with the domain's security realm to secure communication. The SNMP agent decodes SNMP credentials in requests and passes the SNMP username to the security realm. The security realm maps the SNMP username to a WebLogic Server user, authenticates the user, and authorizes access to monitoring data in the domain. To map the SNMP credentials to a user in a WebLogic Server security realm, you create a credential map.

Under Domain Structure, click SNMP. Then click the Security tab. After clicking the New button, select Authentication from the Credential Mapping Type list. In User Name, enter the name of the WebLogic Server user. In SNMP Password, enter the authentication password that SNMP managers will send in their requests. Finally, repeat these steps to create a "Privacy" credential mapping and enter the privacy password that SNMP managers will send in their requests.

To optimize performance, an SNMPv3 agent caches the credential maps that correlate WebLogic Server users with SNMP credentials. To make sure that the cache contains the latest set of SNMP credentials, an agent periodically invalidates its cache. After the cache is invalidated, the next time the agent requests credentials, it regenerates its cache.

Configuring Trap Destination Security



Configuring Trap Destination Security

If the SNMP agent sends SNMPv1 or v2 notifications and the management console requires a specific community to accept traps, use the Community field in the corresponding trap destination in WebLogic Server.

To use a specific SNMPv3 authentication and or privacy protocol when sending responses or notifications, you must also configure the security level of your trap destinations. In Security Name, enter the username on whose behalf the WebLogic SNMP agent sends notifications. The username must be the name of an existing WebLogic Server user for whom you have created an SNMP credential map. When the WebLogic SNMP agent prepares a notification, it uses the credential map to look up and encode SNMP credentials.

In the Security Level list, select a security level that is equal to or higher than the security level that is configured for receiving requests from SNMP managers. For example, if the WebLogic SNMP agent requires incoming SNMPv3 requests to use the authentication protocol, the security level for this trap destination must either require authentication or both authentication and privacy.

WLS SNMP Utility

WLS includes a command-line SNMP utility that supports:

- V1, V2, or V3 messages and all message types
- Polling and capturing traps (UDP or TCP)
- MIB modules (XML format)
- V3 security features

Print Help for a specific command:

```
java weblogic.diagnostics.snmp.cmdline.Manager SnmpWalk -?
```

Poll a management object by using the WLS MIB:

```
java weblogic.diagnostics.snmp.cmdline.Manager SnmpWalk
   -M /weblogic/diagnostics/snmp/mib -m BEA-WEBLOGIC-MIB
   -h localhost -p 7090 -O -u myuser -A mypassword
   -e myEngineID safAgentRuntimeMessagesPendingCount
```


WLS SNMP Utility

WebLogic Server provides a command-line utility that offers many of the same features as an SNMP manager. You can use this utility to test and troubleshoot the configuration of your SNMP agents in a WebLogic Server domain. Like most WLS tools, first open a command prompt (shell) and invoke the following script: <WL_HOME>\server\bin\setWLSEnv.sh (or setWLSEnv.cmd on Windows). The script adds a supported JDK to the shell's PATH environment variable and adds WebLogic Server classes to the CLASSPATH variable.

Available commands include:

`SnmpGet:` Retrieves the value of one or more MIB variables. This command does not accept OIDs for managed objects. You can specify an optional interval at which this command repeatedly retrieves the value of the specified variable.

`SnmpInform:` Constructs a test INFORM notification and distributes it to an SNMP manager or trap monitor.

`SnmpGetBulk:` Returns a collection of MIB variables by repeatedly invoking SnmpGetNext in a pattern that you specify. Use the -Bn and -Bm arguments and one or more OIDs to specify the pattern.

`SnmpTrapMonitor:` Starts a process that listens for notifications. Prints each notification that it receives to standard out.

Section Summary

In this section, you should have learned how to:

- Explain the basic SNMP architecture
- Locate the WLS SNMP MIB
- Compare a centralized and decentralized SNMP implementation
- Configure an SNMP agent
- Define SNMP traps and destinations

Practice 18-2
Monitor WLS Using SNMP

This practice covers the following topics:

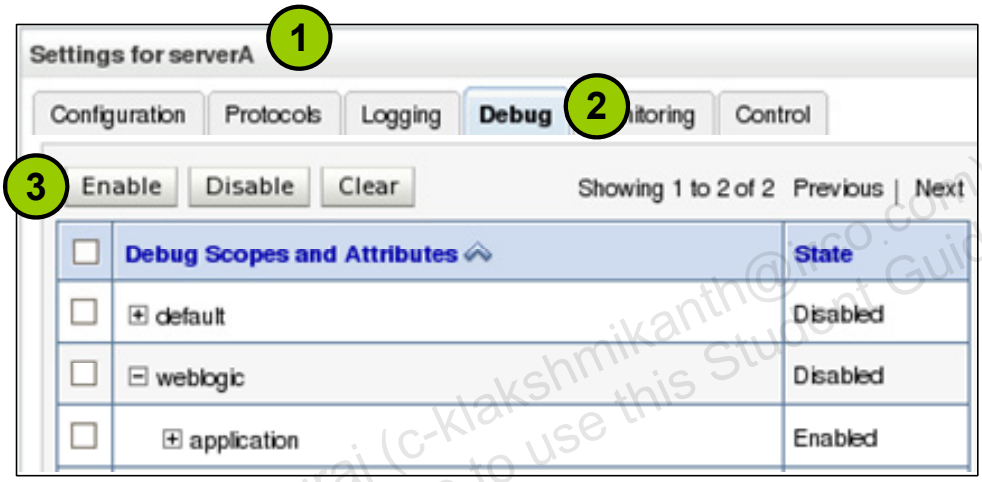
- Mapping WLS and SNMP credentials
- Configuring, deploying, and monitoring SNMP agents
- Configuring an agent's trap destinations
- Using the WLS SNMP command-line utilities
- Creating a custom trap monitor based on MBean attributes

Road Map

- Monitoring Concepts
- WLDF Fundamentals
- WLS SNMP
- WLS Debugging
  - Debug Scopes
  - Debug Logging

Subsystem Debugging

- Various WLS subsystems have the ability to produce very detailed log messages for debugging purposes.
- You can enable debugging on specific servers and for individual subsystems.

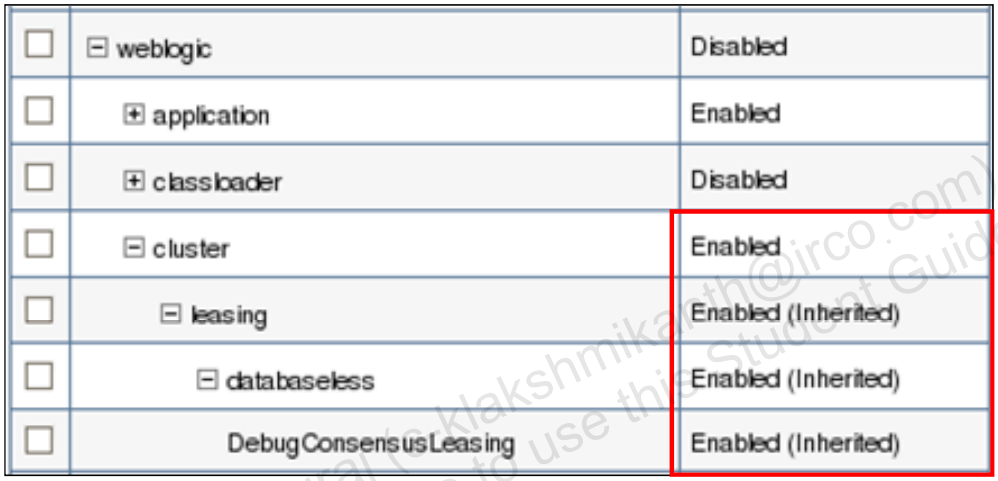


Subsystem Debugging

1. Select an existing server.
2. Click the Debug tab. Use this page to define the debug settings for this server.
3. Select one or more available debugging scopes using the supplied check boxes. Then click Enable or Disable. For convenience, a Clear button is also provided.

Debug Scopes

- Debugging flags for WLS subsystems are classified and organized into scopes.
- When a parent scope is enabled, all child scopes are also enabled, unless overridden.



ORACLE

Debug Scopes

This debugging method is dynamic and can be used to enable debugging while the server is running. Alternatively, many debug flags can also be set as command-line arguments when starting a server. For example:

```
-Dweblogic.debug.DebugJDBCSQL=true
-Dweblogic.debug.DebugJMSBackEnd=true
-Dweblogic.debug.DebugSAFSendingAgent=true
-Dweblogic.debug.DebugJDBCJTA=true
```

Another alternative debugging technique that is specific to JDBC is WebLogic JDBC Spy, which wraps a WebLogic Type 4 JDBC driver. It logs detailed information about JDBC calls issued by an application and then passes the calls to the wrapped WebLogic Type 4 JDBC driver. You can use the information in the logs to help troubleshoot problems in your application. To use WebLogic JDBC Spy with WebLogic Server, you add JDBC Spy attributes to the end of the URL in the JDBC data source configuration. Also add `<WL_HOME>/server/lib/wlspy.jar` to your server's classpath. For example:

```
jdbc:bea:DB2://db2host:50000;spyAttributes=(log=(file)d:\spy.log;
timestamp=yes)
```

Example Debug Scopes and Attributes

Subsystem	Scopes (weblogic.*)
JDBC	jdbc.connection, jdbc.sql, jpa.jdbc, jdbc.internal
JMS	jms.module, store, jms.store, jms.durablesubscribers, messaging.kernel
Message Bridge	messaging bridge
SAF	messaging.saf, jms.saf
Cluster	core.cluster, cluster.leasing
Deployment	deploy, ejb.deployment
Applications	ejb.pooling, ejb.caching, ejb.invoke, application.library
Transactions	transaction.xa, transaction.twopc, transaction.recovery, jms.xa

Example Debug Scopes and Attributes

- **`weblogic.jdbc.sql.DebugJDBCSQL:`** Prints information about all JDBC methods invoked, including their arguments and return values, and thrown exceptions
- **`weblogic.jdbc.connection.DebugJDBCConn:`** Traces all connection reserve and release operations in data sources as well as all application requests to get or close connections
- **`weblogic.jdbc.internal.DebugJDBCInternal:`** Low-level debugging related to the data source, the connection environment, and the data source manager
- **`weblogic.jms.backend.DebugJMSBackEnd:`** Prints information for debugging the JMS Back End (including some information used for distributed destinations and JMS SAF)
- **`weblogic.jms.frontend.DebugJMSFrontEnd:`** Prints information for debugging the JMS Front End (including some information used for multicast)
- **`weblogic.jms.common.DebugJMSCommon:`** Prints information for debugging JMS common methods (including some information from the client JMS producer)
- **`weblogic.jms.boot.DebugJMSBoot:`** Prints some messages at boot time regarding what store the JMS server is using and its configured destinations

Example Debug Scopes and Attributes (continued)

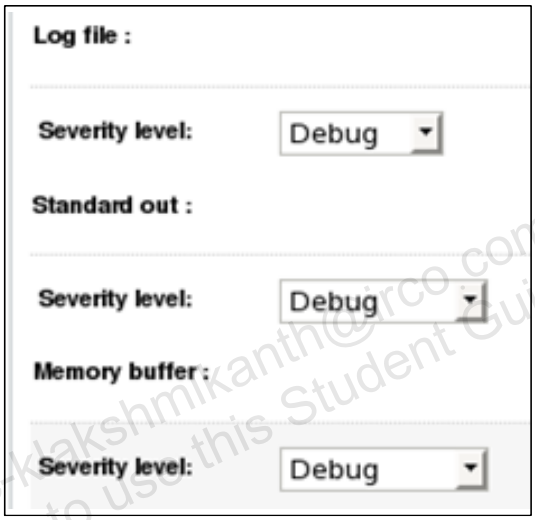
- **`weblogic.jms.module.DebugJMSModule:`** Prints a lot of information about JMS module operations and message life cycle
- **`weblogic.messaging.kernel.DebugMessagingKernel:`** Prints information about the messaging kernel
- **`weblogic.messaging.saf.store.DebugSAFStore:`** Prints limited information about SAF's use of the store
- **`weblogic.messaging.saf.sendingagent.DebugSAFSendingAgent:`** Prints information about the SAF sending side
- **`weblogic.messaging.saf.verbose.DebugSAFVerbose:`** Prints detailed (internal) information about SAF processing
- **`weblogic.jms.saf.DebugJMSSAF:`** Prints information about JMS SAF (store-and-forward) destinations
- **`weblogic.transaction.xa.DebugJTAXA:`** Traces for XA resources
- **`weblogic.transaction.stacktrace.DebugJTAXAStackTrace:`** Detailed tracing that prints stack traces at various critical locations
- **`weblogic.transaction.twopc.DebugJTA2PC:`** Traces all two-phase commit operations
- **`weblogic.transaction.twopcstacktrace.DebugJTA2PCStackTrace:`** Detailed two-phase commit tracing that prints stack traces
- **`weblogic.transaction.tlog.DebugJTATLOG:`** Traces transaction logging information
- **`weblogic.transaction.recovery.DebugJTARecovery:`** Traces recovery information
- **`weblogic.transaction.migration.DebugJTAMigration:`** Traces information about Transaction Log migration
- **`weblogic.transaction.llr.DebugJTALLR:`** Traces all Logging Last Resource operations
- **`weblogic.transaction.health.DebugJTAHealth:`** Traces information about transaction subsystem health

It is also possible to see the tree view of all debug scope definitions using this WLS utility:

`java weblogic.diagnostics.debug.DebugScopeViewer`.

Debug Logging

- By default, debug messages are not written to the server log or standard output.
- Modify the minimum severity of log streams to include debug messages.



Debug Logging

You can enable debugging by setting the appropriate Severity Level server logging configuration attributes to true.

Section Summary

In this section, you should have learned how to:

- Enable debugging for specific WLS features
- Direct debug messages to log destinations

Quiz

Name four tools that can be used to monitor WLS.

a. FMW Control
b. WLST
c. SWSS
d. WLDF
e. NMP

Answer: a, b, d, e

Quiz

What WLST command is necessary to monitor the health or performance of WLS resources?

a. `serverConfig()`
b. `redeploy()`
c. `startEdit()`
d. `serverRuntime()`
e. `validate()`

Answer: d

Quiz

Which of the following is NOT an available action for a WLDF monitor (instrumentation)?

a. Trap Destination
b. Stack Dump
c. Elapsed Time
d. Display Arguments
e. Thread Dump

Answer: a

Quiz

Which of the following is NOT a configuration attribute for WLS SNMP agents?

a. Trap Version
b. Data Source
c. Community Prefix
d. Inform Enabled
e. Engine ID

Answer: b

Lesson Summary

In this lesson, you should have learned how to:

- List several tools for monitoring and troubleshooting WebLogic Server
- Describe the role that JMX plays in WLS monitoring
- Harvest MBean metrics using WLDF
- Configure diagnostic watches and notifications
- Configure and secure the WLS SNMP agent
- Define custom SNMP notifications
- Use server debugging attributes

Practice 18-3
Debug WLS Subsystems

This practice covers the following topics:

- Enabling debugging on a server subsystem
- Configuring server logs to include debug messages
- Interpreting debug messages